cars-eda

January 20, 2023

Exploratory Data Analysis

1 Problem Statement:

We have used Cars dataset from kaggle with features including make, model, year, engine, and other properties of the car used to predict its price.

1.1 Importing the necessary libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)
from scipy import stats
import warnings
warnings.filterwarnings("ignore")
```

1.2 Load the dataset into dataframe

[2]:
```
## load the csv file
df = pd.read_csv('/content/Cars_data.csv')
```

[3]:
```
## print the head of the dataframe
df.head()
```

[3]:
```
  Make       Model  Year             Engine Fuel Type  Engine HP  \
0  BMW  1 Series M  2011  premium unleaded (required)      335.0
1  BMW    1 Series  2011  premium unleaded (required)      300.0
2  BMW    1 Series  2011  premium unleaded (required)      300.0
3  BMW    1 Series  2011  premium unleaded (required)      230.0
4  BMW    1 Series  2011  premium unleaded (required)      230.0

   Engine Cylinders Transmission Type     Driven_Wheels  Number of Doors  \
0               6.0            MANUAL  rear wheel drive              2.0
1               6.0            MANUAL  rear wheel drive              2.0
```

```
2            6.0         MANUAL  rear wheel drive            2.0
3            6.0         MANUAL  rear wheel drive            2.0
4            6.0         MANUAL  rear wheel drive            2.0

                   Market Category Vehicle Size Vehicle Style  \
0  Factory Tuner,Luxury,High-Performance      Compact         Coupe
1                     Luxury,Performance      Compact   Convertible
2                Luxury,High-Performance      Compact         Coupe
3                     Luxury,Performance      Compact         Coupe
4                                 Luxury      Compact   Convertible

   highway MPG  city mpg  Popularity   MSRP
0           26        19        3916  46135
1           28        19        3916  40650
2           28        20        3916  36350
3           28        18        3916  29450
4           28        18        3916  34500
```

Now we observe the each features present in the dataset.

`Make:` The Make feature is the company name of the Car. `Model:` The Model feature is the model or different version of Car models. `Year:` The year describes the model has been launched. `Engine Fuel Type:` It defines the Fuel type of the car model. `Engine HP:` It's say the Horsepower that refers to the power an engine produces. `Engine Cylinders:` It define the nos of cylinders in present in the engine. `Transmission Type:` It is the type of feature that describe about the car transmission type i.e Mannual or automatic. `Driven_Wheels:` The type of wheel drive. `No of doors:` It defined nos of doors present in the car. `Market Category:` This features tells about the type of car or which category the car belongs. `Vehicle Size:` It's say about the about car size. `Vehicle Style:` The feature is all about the style that belongs to car. `highway MPG:` The average a car will get while driving on an open stretch of road without stopping or starting, typically at a higher speed. `city mpg:` City MPG refers to driving with occasional stopping and braking. `Popularity:` It can refered to rating of that car or popularity of car. `MSRP:` The price of that car.

1.3 Check the datatypes

[4]:
```python
# Get the datatypes of each columns number of records in each column.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Make               11914 non-null  object
 1   Model              11914 non-null  object
 2   Year               11914 non-null  int64
 3   Engine Fuel Type   11911 non-null  object
```

```
 4   Engine HP          11845 non-null  float64
 5   Engine Cylinders   11884 non-null  float64
 6   Transmission Type  11914 non-null  object
 7   Driven_Wheels      11914 non-null  object
 8   Number of Doors    11908 non-null  float64
 9   Market Category    8172 non-null   object
 10  Vehicle Size       11914 non-null  object
 11  Vehicle Style      11914 non-null  object
 12  highway MPG        11914 non-null  int64
 13  city mpg           11914 non-null  int64
 14  Popularity         11914 non-null  int64
 15  MSRP               11914 non-null  int64
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

1.4 Dropping irrevalent columns

If we consider all columns present in the dataset then unneccessary columns will impact on the model's accuracy. Not all the columns are important to us in the given dataframe, and hence we would drop the columns that are irrevalent to us. It would reflect our model's accucary so we need to drop them. Otherwise it will affect our model.

The list cols_to_drop contains the names of the cols that are irrevalent, drop all these cols from the dataframe.

`cols_to_drop = ["Engine Fuel Type", "Market Category", "Vehicle Style", "Popularity", "Number of Doors", "Vehicle Size"]`

These features are not neccessary to obtain the model's accucary. It does not contain any relevant information in the dataset.

[5]:
```python
# initialise cols_to_drop
cols_to_drop = ["Engine Fuel Type", "Market Category", "Vehicle Style",␣
 ↪"Popularity", "Number of Doors", "Vehicle Size"]
```

[6]:
```python
# drop the irrevalent cols and print the head of the dataframe
df = df.drop(cols_to_drop,axis=1)

# print df head
df.head()
```

[6]:
```
  Make       Model  Year  Engine HP  Engine Cylinders Transmission Type  \
0  BMW  1 Series M  2011      335.0               6.0            MANUAL
1  BMW    1 Series  2011      300.0               6.0            MANUAL
2  BMW    1 Series  2011      300.0               6.0            MANUAL
3  BMW    1 Series  2011      230.0               6.0            MANUAL
4  BMW    1 Series  2011      230.0               6.0            MANUAL

      Driven_Wheels  highway MPG  city mpg   MSRP
```

```
0  rear wheel drive           26         19  46135
1  rear wheel drive           28         19  40650
2  rear wheel drive           28         20  36350
3  rear wheel drive           28         18  29450
4  rear wheel drive           28         18  34500
```

1.5 Renaming the columns

Now, Its time for renaming the feature to useful feature name. It will help to use them in model training purpose.

We have already dropped the unneccesary columns, and now we are left with useful columns. One extra thing that we would do is to rename the columns such that the name clearly represents the essence of the column.

The given dict represents (in key value pair) the previous name, and the new name for the dataframe columns

[7]:
```
# rename cols
rename_cols = {'Make':'Company_Name','Engine HP':'HP','Engine Cylinders':
↪'Cylinders','MSRP':'Price'}
```

[8]:
```
# use a pandas function to rename the current columns -
df = df.rename(columns=rename_cols)
```

[9]:
```
# Print the head of the dataframe

df.head()
```

[9]:
```
  Company_Name       Model  Year     HP  Cylinders Transmission Type  \
0          BMW  1 Series M  2011  335.0        6.0            MANUAL
1          BMW    1 Series  2011  300.0        6.0            MANUAL
2          BMW    1 Series  2011  300.0        6.0            MANUAL
3          BMW    1 Series  2011  230.0        6.0            MANUAL
4          BMW    1 Series  2011  230.0        6.0            MANUAL

      Driven_Wheels  highway MPG  city mpg  Price
0  rear wheel drive           26        19  46135
1  rear wheel drive           28        19  40650
2  rear wheel drive           28        20  36350
3  rear wheel drive           28        18  29450
4  rear wheel drive           28        18  34500
```

1.6 Dropping the duplicate rows

There are many rows in the dataframe which are duplicate, and hence they are just repeating the information. Its better if we remove these rows as they don't add any value to the dataframe.

For given data, we would like to see how many rows were duplicates. For this, we will count the number of rows, remove the dublicated rows, and again count the number of rows.

[10]:
```
# number of rows before removing duplicated rows

df.shape
#no. of rows=11914.
```

[10]:
```
(11914, 10)
```

[11]:
```
# drop the duplicated rows
df = df.drop_duplicates()

# print head of df
df.head()
```

[11]:
```
  Company_Name         Model  Year     HP  Cylinders Transmission Type  \
0          BMW  1 Series M  2011  335.0        6.0            MANUAL
1          BMW    1 Series  2011  300.0        6.0            MANUAL
2          BMW    1 Series  2011  300.0        6.0            MANUAL
3          BMW    1 Series  2011  230.0        6.0            MANUAL
4          BMW    1 Series  2011  230.0        6.0            MANUAL

      Driven_Wheels  highway MPG  city mpg  Price
0  rear wheel drive           26        19  46135
1  rear wheel drive           28        19  40650
2  rear wheel drive           28        20  36350
3  rear wheel drive           28        18  29450
4  rear wheel drive           28        18  34500
```

[12]:
```
# Count Number of rows after deleting duplicated rows

df.shape   #now no. of rows=10925.
```

[12]:
```
(10925, 10)
```

[13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10925 entries, 0 to 11913
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Company_Name       10925 non-null  object
 1   Model              10925 non-null  object
 2   Year               10925 non-null  int64
 3   HP                 10856 non-null  float64
 4   Cylinders          10895 non-null  float64
```

```
 5   Transmission Type  10925 non-null  object
 6   Driven_Wheels      10925 non-null  object
 7   highway MPG        10925 non-null  int64
 8   city mpg           10925 non-null  int64
 9   Price              10925 non-null  int64
dtypes: float64(2), int64(4), object(4)
memory usage: 938.9+ KB
```

1.7 Dropping the null or missing values

Missing values are usually represented in the form of Nan or null or None in the dataset.

Finding whether we have null values in the data is by using the isnull() function.

There are many values which are missing, in pandas dataframe these values are reffered to as np.nan. We want to deal with these values beause we can't use nan values to train models. Either we can remove them to apply some strategy to replace them with other values.

To keep things simple we will be dropping nan values

[14]:
```
# check for nan values in each columns

df.isnull().sum()
```

[14]:
```
Company_Name          0
Model                 0
Year                  0
HP                   69
Cylinders            30
Transmission Type     0
Driven_Wheels         0
highway MPG           0
city mpg              0
Price                 0
dtype: int64
```

As we can see that the HP and Cylinders have null values of 69 and 30. As these null values will impact on models' accuracy. So to avoid the impact we will drop the these values. As these values are small camparing with dataset that will not impact any major affect on model accuracy so we will drop the values.

[15]:
```
# drop missing values
df = df.dropna(how='any')
```

[16]:
```
# Make sure that missing values are removed
# check number of nan values in each col again

df.isnull().sum()
```

[16]:
```
Company_Name         0
Model                0
Year                 0
HP                   0
Cylinders            0
Transmission Type    0
Driven_Wheels        0
highway MPG          0
city mpg             0
Price                0
dtype: int64
```

[17]:
```
#Describe statistics of df
df.describe()
```

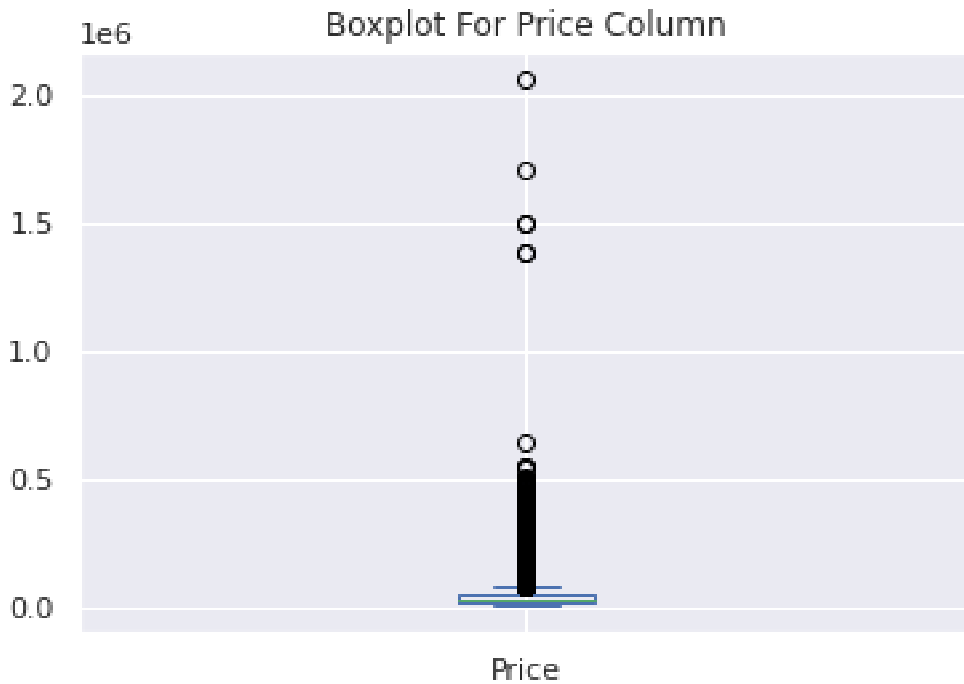
[17]:
```
               Year            HP     Cylinders   highway MPG      city mpg  \
count  10827.000000  10827.000000  10827.000000  10827.000000  10827.000000
mean    2010.896370    254.553062      5.691604     26.308119     19.327607
std        7.029534    109.841537      1.768551      7.504652      6.643567
min     1990.000000     55.000000      0.000000     12.000000      7.000000
25%     2007.000000    173.000000      4.000000     22.000000     16.000000
50%     2015.000000    240.000000      6.000000     25.000000     18.000000
75%     2016.000000    303.000000      6.000000     30.000000     22.000000
max     2017.000000   1001.000000     16.000000    354.000000    137.000000

              Price
count  1.082700e+04
mean   4.249325e+04
std    6.229451e+04
min    2.000000e+03
25%    2.197250e+04
50%    3.084500e+04
75%    4.330000e+04
max    2.065902e+06
```

1.8 Removing outliers

Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called outliers and often machine learning modeling and model skill in general can be improved by understanding and even removing these outlier values.

[18]:
```
## Plot a boxplot for 'Price' column in dataset.
df['Price'].plot(kind='box',title='Boxplot For Price Column')
#sns.boxplot(df['Price'])
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f967d709940>
```

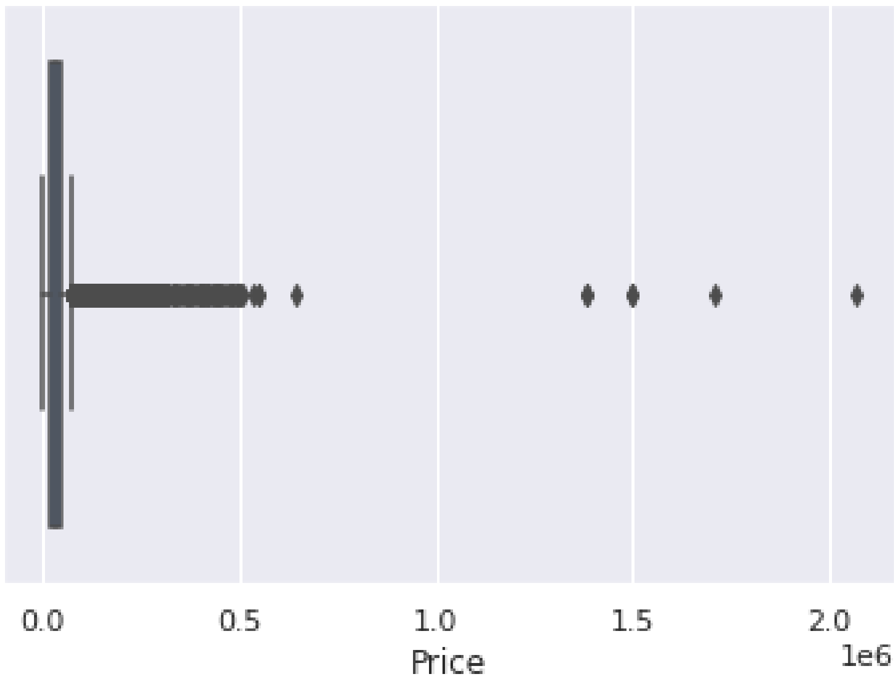


[19]:
```
df.boxplot('Price')
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f967d6bc7c0>

[20]:
```
sns.boxplot(df['Price'])
```

[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f967d1c1040>
```

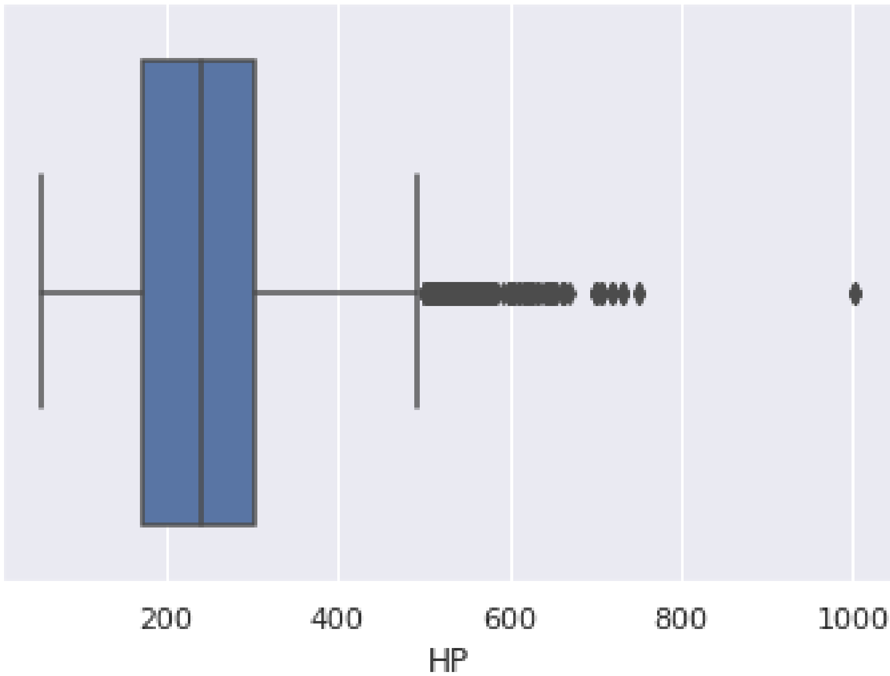


1.8.1 Observation:

Here as you see that we got some values near to 1.5 and 2.0 . So these values are called outliers. Because there are away from the normal values. Now we have detect the outliers of the feature of Price. Similarly we will checking of anothers features.

[21]:
```
## PLot a boxplot for 'HP' columns in dataset
sns.boxplot(df['HP'])
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f967d168220>
```



1.8.2 Observation:

Here boxplots show the proper distribution of of 25 percentile and 75 percentile of the feature of HP.

[22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10827 entries, 0 to 11913
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Company_Name       10827 non-null  object
 1   Model              10827 non-null  object
 2   Year               10827 non-null  int64
 3   HP                 10827 non-null  float64
 4   Cylinders          10827 non-null  float64
 5   Transmission Type  10827 non-null  object
 6   Driven_Wheels      10827 non-null  object
 7   highway MPG        10827 non-null  int64
 8   city mpg           10827 non-null  int64
 9   Price              10827 non-null  int64
dtypes: float64(2), int64(4), object(4)
memory usage: 930.4+ KB
```

print all the columns which are of int or float datatype in df.

Hint: Use loc with condition

[23]:
```
# print all the columns which are of int or float datatype in df.

int_or_float_columns=df.select_dtypes(include=['int64','float64']).columns
```

[24]:
```
int_or_float_columns
```

[24]:
```
Index(['Year', 'HP', 'Cylinders', 'highway MPG', 'city mpg', 'Price'],
dtype='object')
```

1.8.3 Save the column names of the above output in variable list named 'l'

[25]:
```
# save column names of the above output in variable list
l=list(int_or_float_columns)
l
```

[25]:
```
['Year', 'HP', 'Cylinders', 'highway MPG', 'city mpg', 'Price']
```

1.9 Outliers removal techniques - IQR Method

Here comes cool Fact for you!

IQR is the first quartile subtracted from the third quartile; these quartiles can be clearly seen on a box plot on the data.

- Calculate IQR and give a suitable threshold to remove the outliers and save this new dataframe into df2.

Let us help you to decide threshold: Outliers in this case are defined as the observations that are below (Q1 − 1.5x IQR) or above (Q3 + 1.5x IQR)

[26]:
```
## define Q1 and Q2
Q1 = np.percentile(df[l],25,interpolation='midpoint')
Q3 = np.percentile(df[l],75,interpolation='midpoint')

# # define IQR (interquantile range)
IQR = Q3-Q1

#upper bound
upper=np.where(df[l]>=(Q3+1.5*IQR))
#lower bound
lower=np.where(df[l]<=(Q1-1.5*IQR))

# # define df2 after removing outliers
# df2=df
# df2=df2.drop(lower[0],inplace=True)
```

```
# df2=df2.drop(upper[0],inplace=True)
# df[l]
# df[lower[0]]
def remove_outlier_IQR(df):
    Q1=df.quantile(0.25)
    Q3=df.quantile(0.75)
    IQR=Q3-Q1
    df2=df[~((df<(Q1-1.5*IQR)) | (df>(Q3+1.5*IQR)))]
    return df2
df2=remove_outlier_IQR(df)
# df_outlier_removed=remove_outlier_IQR(df['Price'])
# df_outlier_removed=pd.DataFrame(df_outlier_removed)
# ind_diff=df.index.difference(df_outlier_removed.index)
```

[27]:
```
# len(ind_diff)
```

[28]:
```
df['Price'].shape,df2['Price'].shape
```

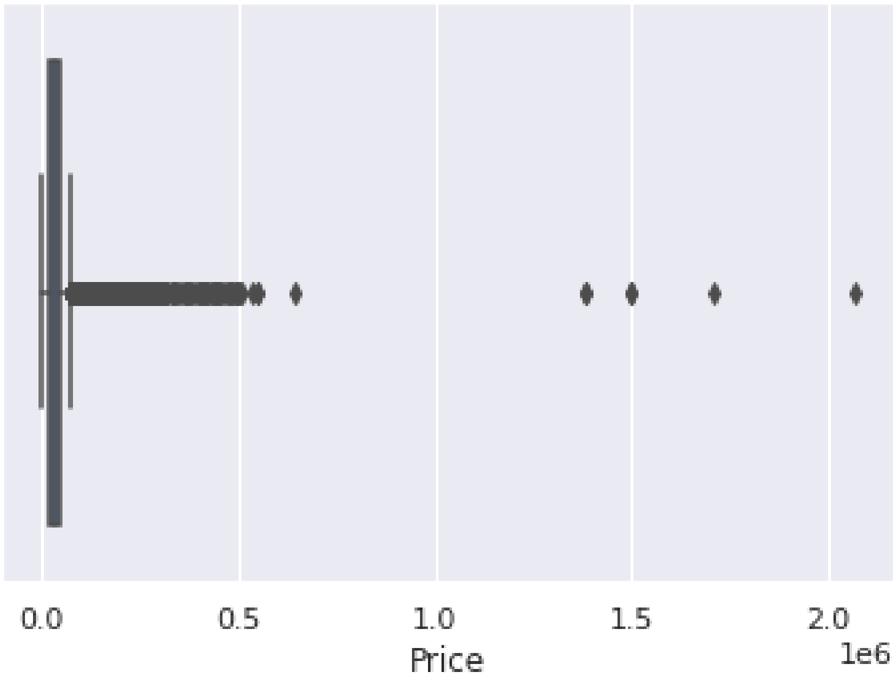
[28]:
```
((10827,), (10827,))
```

[29]:
```
# find the shape of df & df2
print(df.shape,df2.shape)
```

```
(10827, 10) (10827, 10)
```

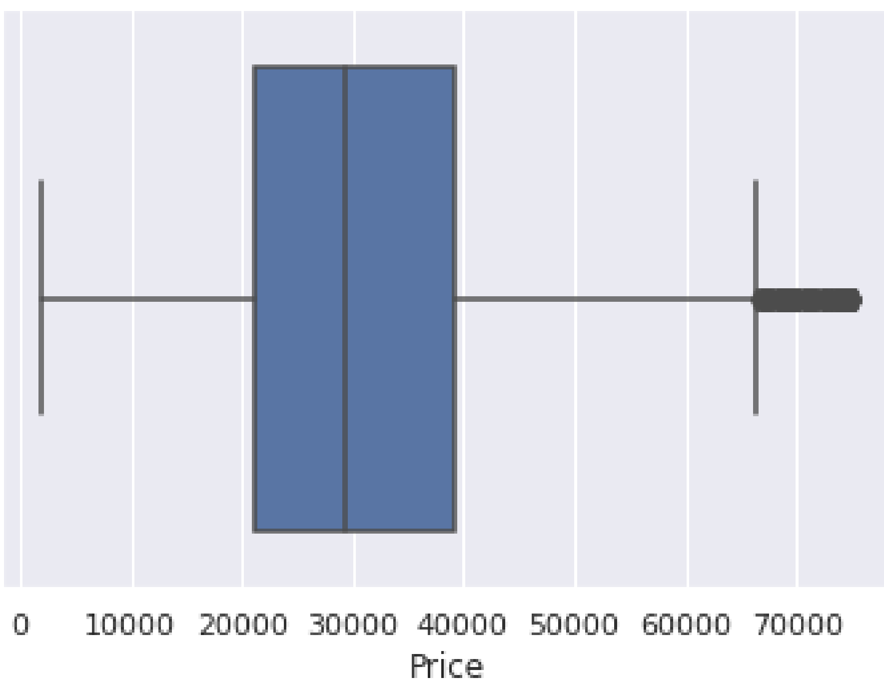
[30]:
```
sns.boxplot(df['Price'])
```

[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f967d791130>
```



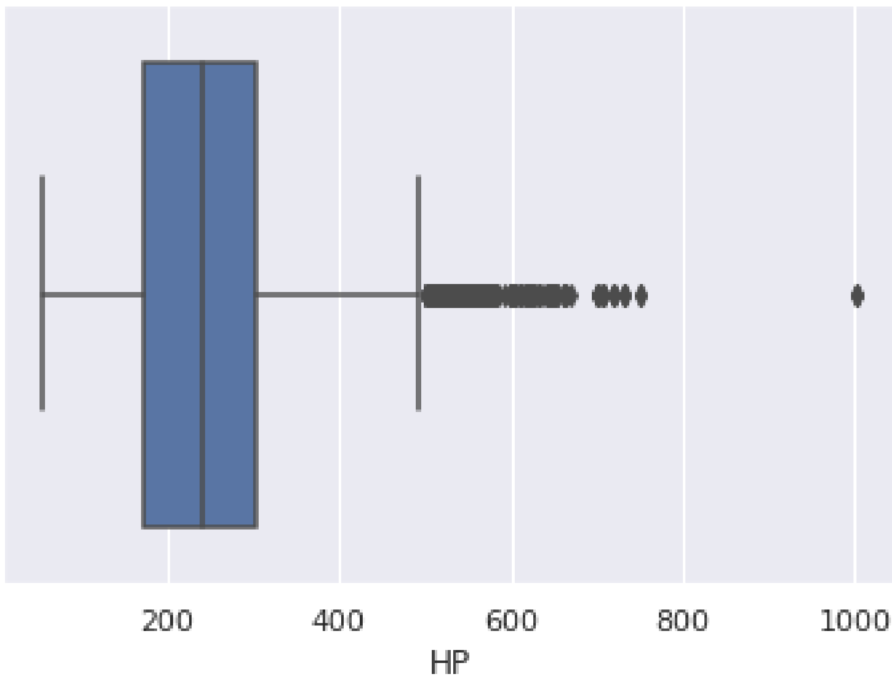

```
[31]: sns.boxplot(df2['Price'])
```

```
[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f968c73a070>
```



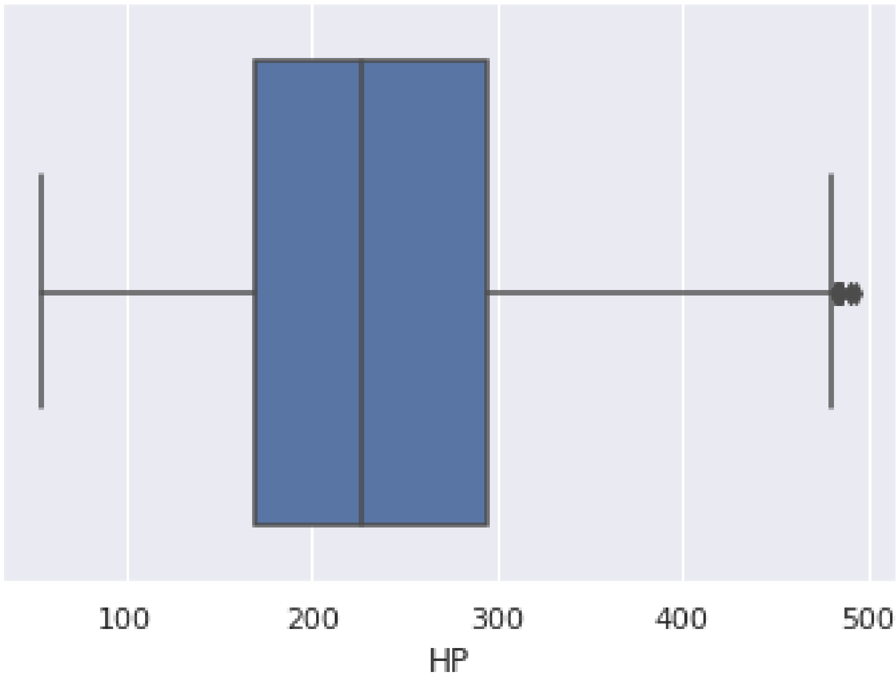
[32]:
```
sns.boxplot(df['HP'])
```

[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f967d162af0>



[33]:
```
sns.boxplot(df2['HP'])
```

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f967cfc4520>



[34]:
```
# find unique values and there counts in each column in df using value counts␣
↪function.

for i in df.columns:
  print ("--------------- %s ----------------" % i)
  print(df[(i)].value_counts())
```

```
--------------- Company_Name ----------------
Chevrolet        1043
Ford              798
Toyota            651
Volkswagen        563
Nissan            540
Dodge             513
GMC               475
Honda             429
Cadillac          396
Mazda             392
Mercedes-Benz     340
Suzuki            338
Infiniti          326
BMW               324
Audi              320
Hyundai           254
```

```
Acura            246
Volvo            241
Subaru           229
Kia              219
Mitsubishi       202
Lexus            201
Chrysler         185
Buick            184
Pontiac          163
Lincoln          152
Porsche          134
Land Rover       126
Oldsmobile       111
Saab             101
Aston Martin      91
Bentley           74
Ferrari           69
Plymouth          62
Scion             60
FIAT              58
Maserati          55
Lamborghini       52
Rolls-Royce       31
Lotus             28
HUMMER            17
Maybach           16
McLaren            5
Alfa Romeo         5
Genesis            3
Bugatti            3
Spyker             2
Name: Company_Name, dtype: int64
--------------- Model ----------------
Silverado 1500   156
F-150            126
Sierra 1500       90
Tundra            78
Frontier          76
                 …
M4 GTS             1
LFA                1
Horizon            1
GS F               1
Zephyr             1
Name: Model, Length: 904, dtype: int64
--------------- Year ----------------
2015    2029
2016    2022
```

```
2017    1580
2014     530
2012     350
2009     349
2007     332
2013     320
2008     316
2011     278
2010     272
2003     233
2004     230
2005     205
2002     203
2006     194
2001     168
1997     148
1998     143
1993     135
2000     114
1999     111
1994     109
1992     104
1995     103
1996      98
1991      84
1990      67
Name: Year, dtype: int64
--------------- HP ----------------
200.0    373
170.0    255
240.0    248
285.0    246
210.0    243
        …
557.0      1
361.0      1
456.0      1
661.0      1
151.0      1
Name: HP, Length: 355, dtype: int64
--------------- Cylinders ----------------
4.0     4227
6.0     4215
8.0     1889
12.0     228
5.0      159
10.0      65
3.0       28
```

```
0.0       13
16.0       3
Name: Cylinders, dtype: int64
--------------- Transmission Type ----------------
AUTOMATIC           7750
MANUAL              2498
AUTOMATED_MANUAL     553
DIRECT_DRIVE          15
UNKNOWN               11
Name: Transmission Type, dtype: int64
--------------- Driven_Wheels ----------------
front wheel drive    4168
rear wheel drive     3120
all wheel drive      2281
four wheel drive     1258
Name: Driven_Wheels, dtype: int64
--------------- highway MPG ----------------
24     822
23     758
26     725
22     686
25     685
28     651
27     555
30     499
21     488
19     488
31     488
20     469
29     425
18     345
17     340
33     329
32     292
34     270
16     199
35     199
36     191
37     166
38     130
15     116
40     109
39     107
41      65
42      46
14      37
43      21
46      21
```

```
44       21
48       16
45       14
13       13
50       10
47        7
109       6
12        5
53        5
82        3
111       3
354       1
106       1
Name: highway MPG, dtype: int64
--------------- city mpg ----------------
17     1154
16     1014
15      949
18      938
19      793
20      742
14      603
22      571
21      551
13      537
23      425
25      392
24      372
12      282
27      243
26      207
11      187
28      160
30      127
31      116
29       98
10       76
9        33
32       21
34       20
36       20
40       19
44       18
42       17
41       17
35       15
33       13
53       13
```

```
43       13
54       10
8         9
37        8
39        6
51        6
50        6
128       6
49        4
137       3
85        3
55        3
47        2
58        2
129       1
7         1
38        1
Name: city mpg, dtype: int64
--------------- Price ----------------
2000     599
29995     18
25995     16
20995     15
27995     15
        …
66347      1
62860      1
48936      1
68996      1
50920      1
Name: Price, Length: 6014, dtype: int64
```

1.10 Visualising Univariate Distributions

We will use seaborn library to visualize eye catchy univariate plots.

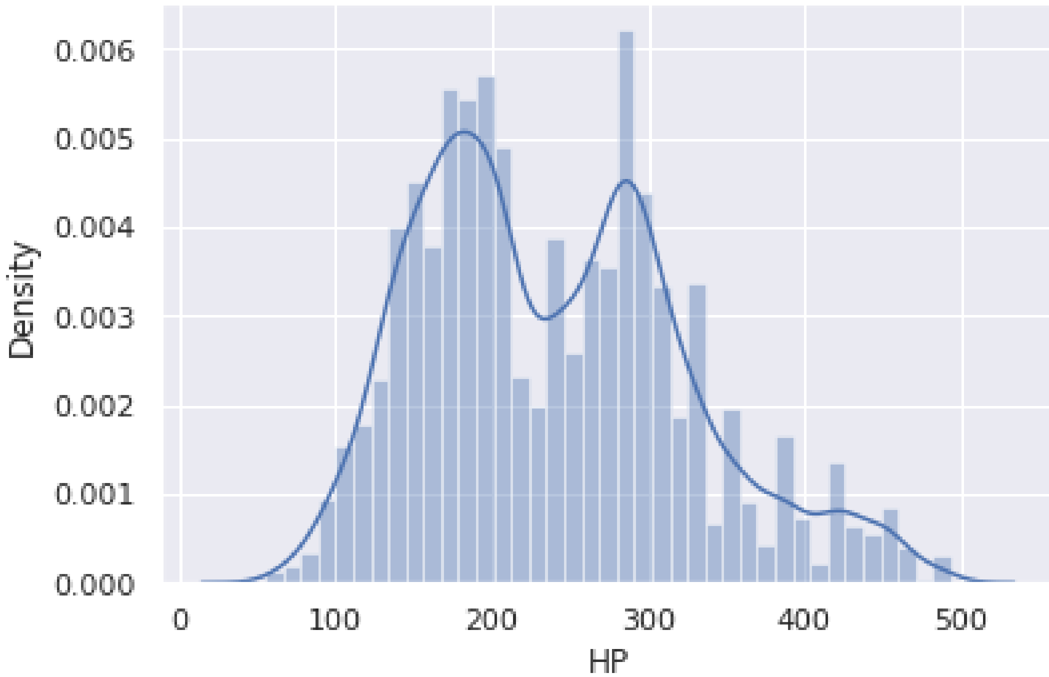
Do you know? you have just now already explored one univariate plot. guess which one? Yeah its box plot.

1.10.1 Histogram & Density Plots

Histograms and density plots show the frequency of a numeric variable along the y-axis, and the value along the x-axis. The `sns.distplot()` function plots a density curve. Notice that this is aesthetically better than vanilla `matplotlib`.

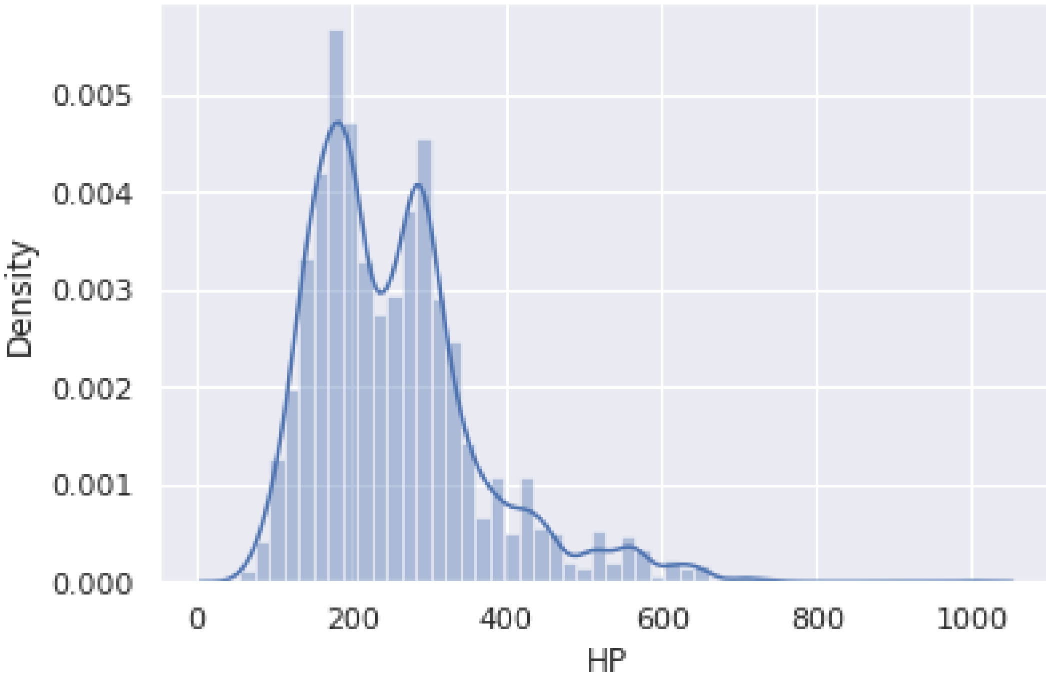
[35]:
```
#ploting distplot for variable HP

sns.distplot(df2['HP'])
```

[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f967cfbffd0>



[36]:
```
sns.distplot(df['HP']) #df contains outliers.
```

[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f967ceebfa0>



1.10.2 Observation:

We plot the Histogram of feature HP with help of distplot in seaborn. In this graph we can see that there is max values near at 200. similary we have also the 2nd highest value near 400 and so on. It represents the overall distribution of continuous data variables.

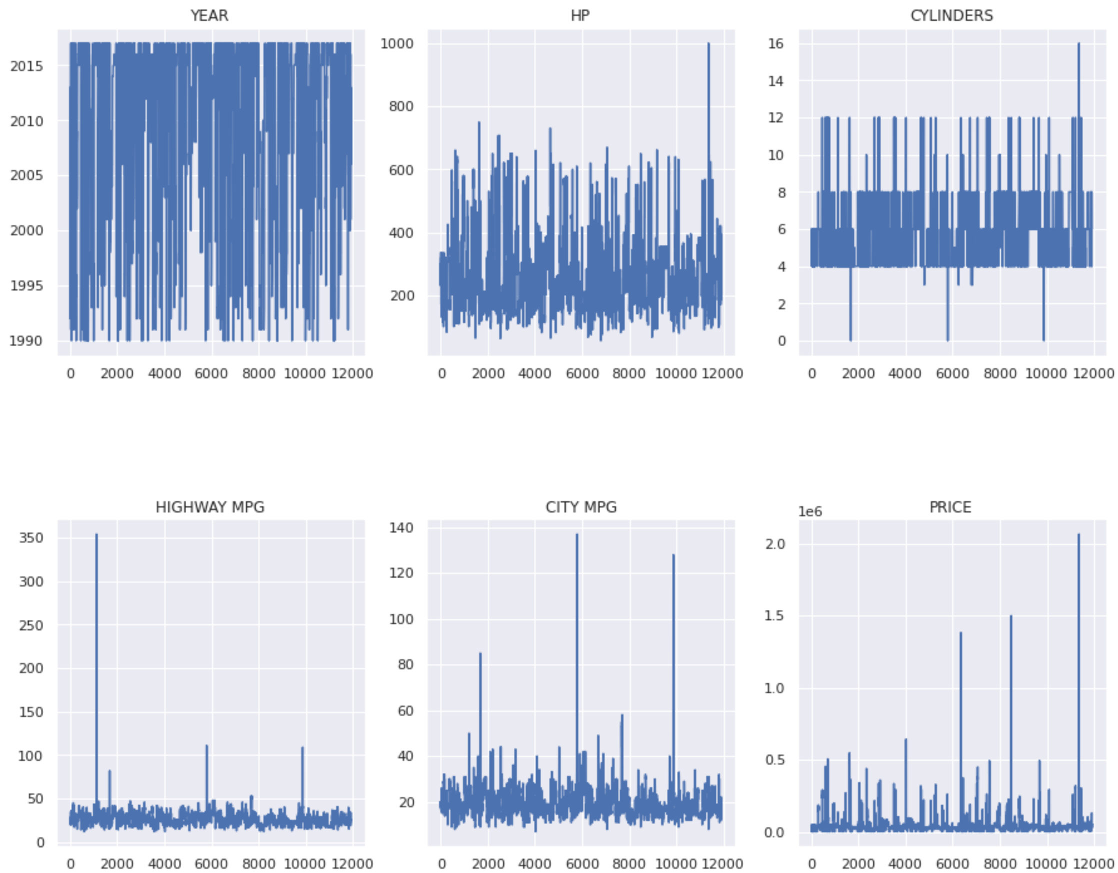
Since seaborn uses matplotlib behind the scenes, the usual matplotlib functions work well with seaborn. For example, you can use subplots to plot multiple univariate distributions. - Hint: use matplotlib subplot function

[37]:
```
l=list(int_or_float_columns)
l
```

[37]: ['Year', 'HP', 'Cylinders', 'highway MPG', 'city mpg', 'Price']

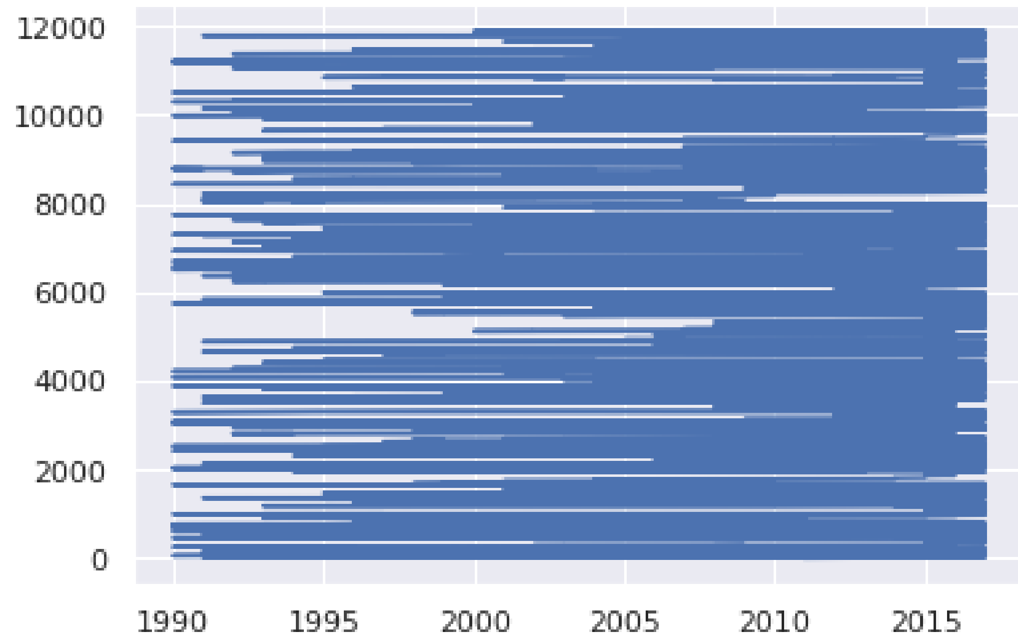
[38]:
```
# plot all the columns present in list l together using subplot of dimention␣
↪(2,3).
# sns.pairplot(df2[l])
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 12))
plt.subplots_adjust(hspace=0.5)

# loop through tickers and axes
for i, ax in zip(l, axs.ravel()):
    # filter df for ticker and plot on specified axes
    df[i].plot(ax=ax)
```

```
    # chart formatting
    ax.set_title(i.upper())
    # ax.get_legend().remove()
    ax.set_xlabel("")

plt.show()
```



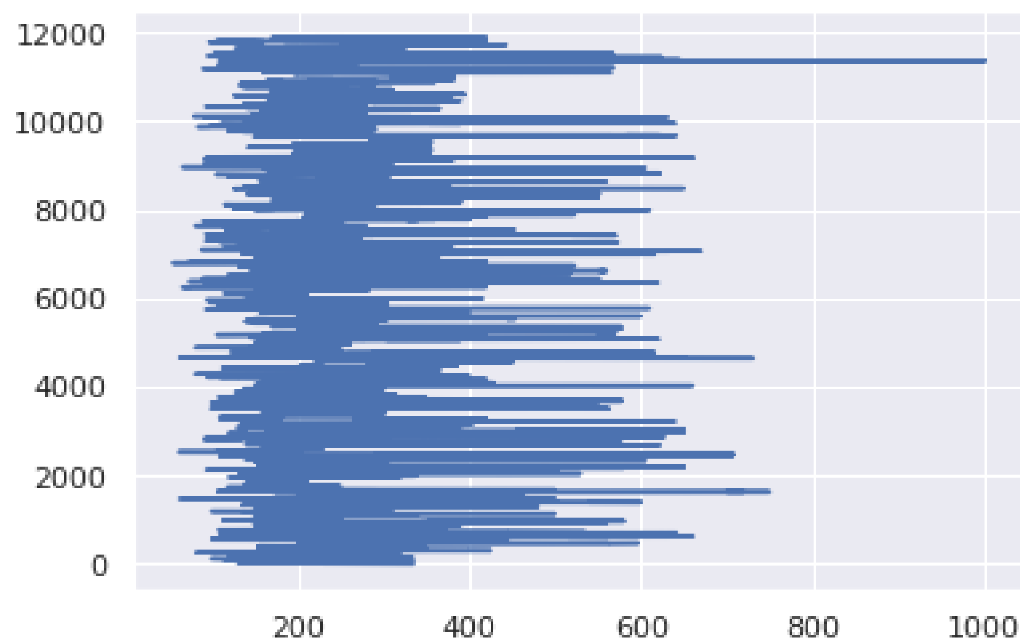
[39]:
```
plt.plot(df['Year'],df['Year'].index)
```

[39]: [<matplotlib.lines.Line2D at 0x7f96785c3700>]



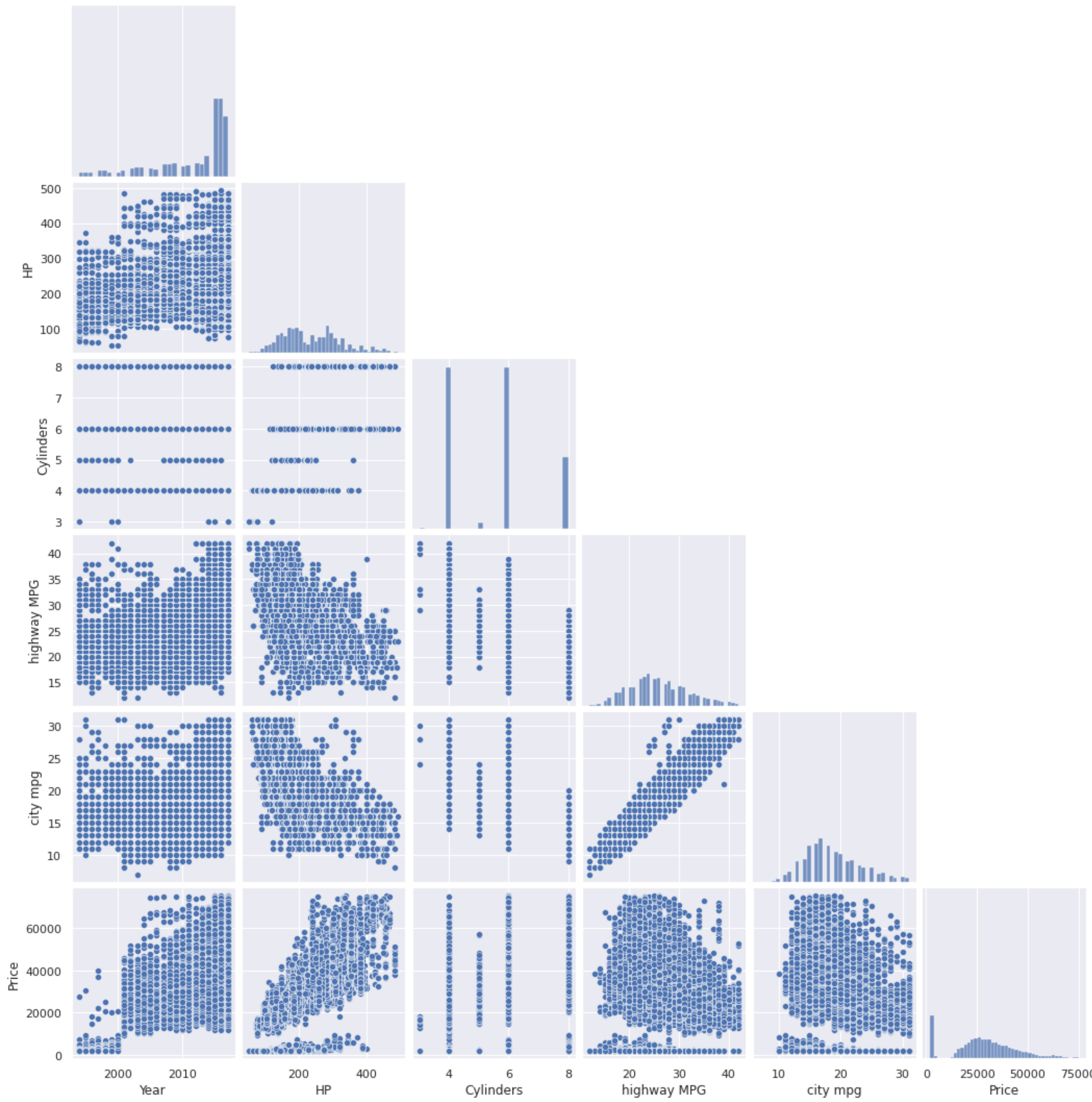
[40]:
```
plt.plot(df['HP'],df['HP'].index)
```

[40]: [<matplotlib.lines.Line2D at 0x7f967842c130>]



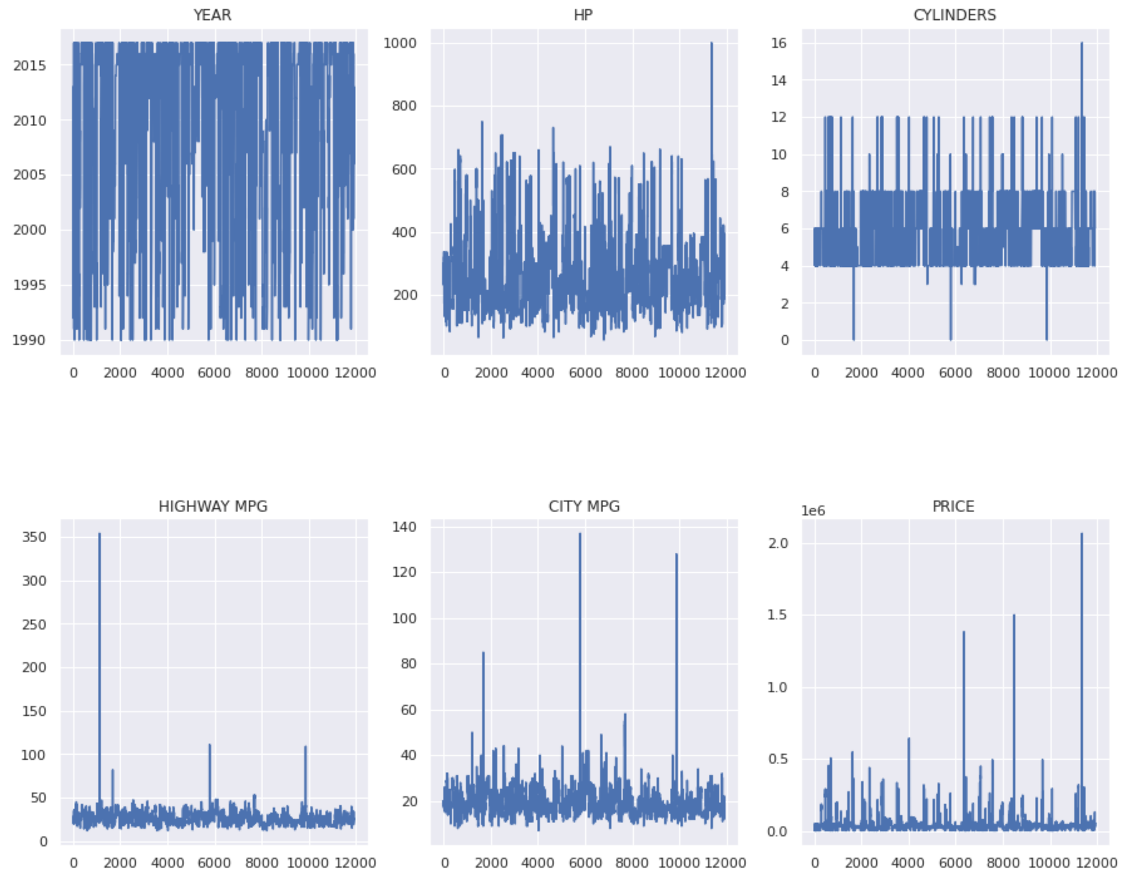
[41]:
```
sns.pairplot(df2[l],corner=True)
```

[41]: <seaborn.axisgrid.PairGrid at 0x7f967cfc4b20>



[42]:
```
plt.figure(figsize=(15, 12))
plt.subplots_adjust(hspace=0.5)

# loop through the length of tickers and keep track of index
for n, ticker in enumerate(l):
    # add a new subplot iteratively
    ax = plt.subplot(2, 3, n + 1)

    # filter df and plot ticker on the new subplot axis
```

```
    df[ticker].plot(ax=ax)

    # chart formatting
    ax.set_title(ticker.upper())
    # ax.get_legend().remove()
    ax.set_xlabel("")
```

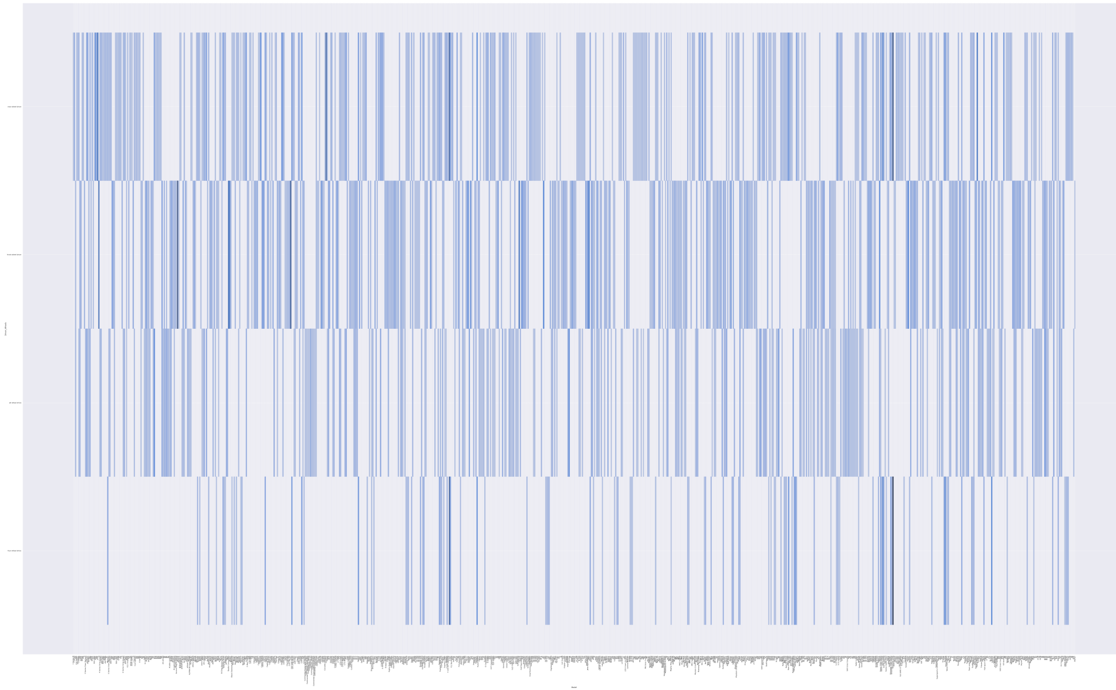


1.11 Bar Chart Plots

Plot a histogram depicting the make in X axis and number of cars in y axis.

[102]:
```
plt.figure(figsize = (132,80))
plt.xticks(rotation=90)
# use nlargest and then .plot to get bar plot like below output
# Plot Title, X & Y label
sns.histplot(data=df2,y=df2['Driven_Wheels'],x=df2['Model'])
# plt.hist([df2['Driven_Wheels'],df2['Model']], rwidth = 0.8, histtype = 'bar',␣
 ↪color = ['violet'])
```

[102]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9654394430>
```



1.11.1 Observation:

In this plot we can see that we have plot the bar plot with the cars model and nos. of cars.

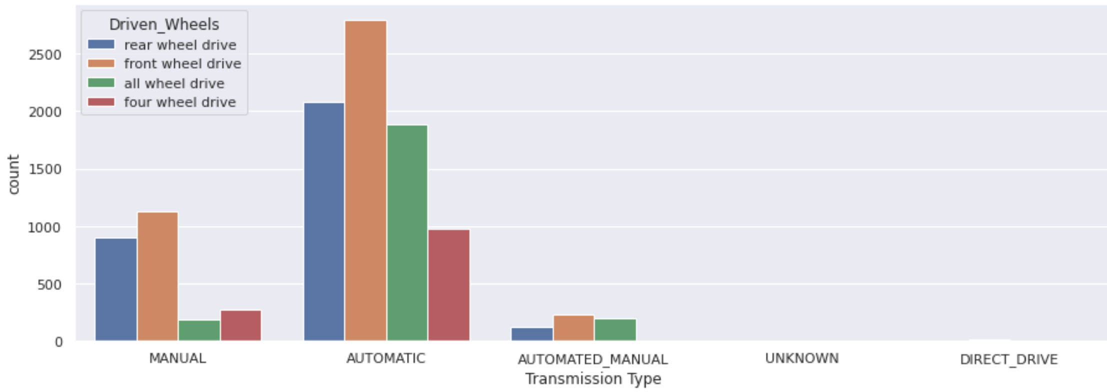
1.11.2 Count Plot

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable.

Plot a countplot for a variable Transmission vertically with hue as Drive mode

[80]:
```
plt.figure(figsize=(15,5))

# plot countplot on transmission and drive mode
# plt.xticks(rotation=90)
sns.countplot(x=df['Transmission Type'],hue='Driven_Wheels',data=df2)
# df.columns
```

[80]: <matplotlib.axes._subplots.AxesSubplot at 0x7f966a6a2820>



1.11.3 Observation:

In this count plot, We have plot the feature of Transmission with help of hue. We can see that the the nos of count and the transmission type and automated manual is plotted. Drive mode as been given with help of hue.

2 Visualising Bivariate Distributions

Bivariate distributions are simply two univariate distributions plotted on x and y axes respectively. They help you observe the relationship between the two variables.

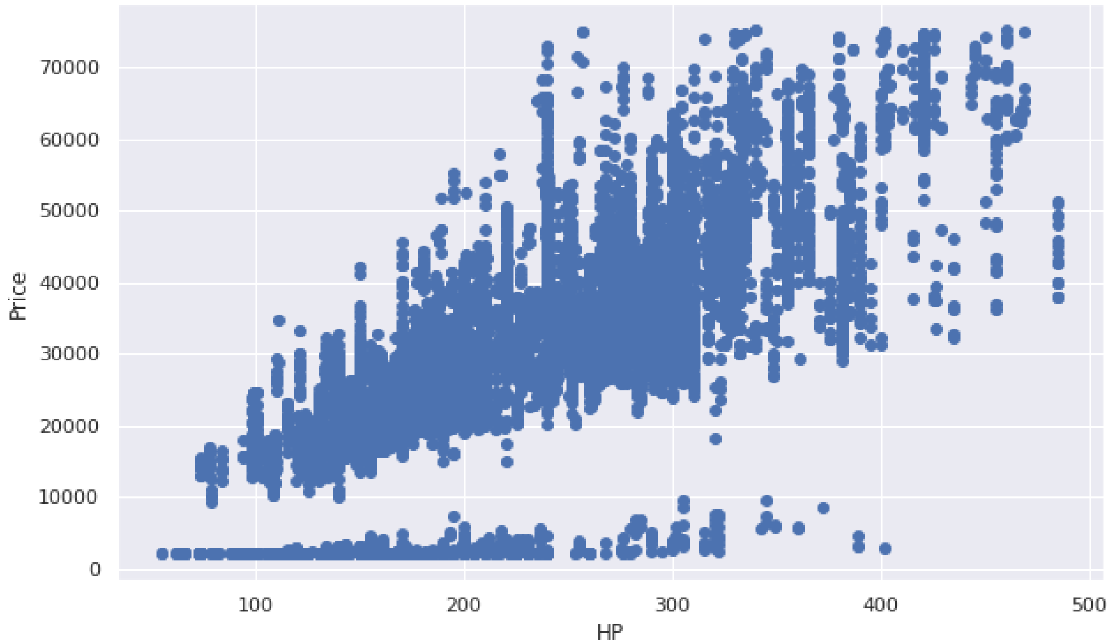
2.1 Scatter Plots

Scatterplots are used to find the correlation between two continuos variables.

Using scatterplot find the correlation between 'HP' and 'Price' column of the data.

[68]:
```
## Your code here -
fig, ax = plt.subplots(figsize=(10,6))

# plot scatterplot on hp and price
plt.scatter(df2['HP'],df2['Price'])
plt.xlabel('HP')
plt.ylabel('Price')
```

[68]:
```
Text(0, 0.5, 'Price')
```



2.1.1 Observation:

It is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. We have plot the scatter plot with x axis as HP and y axis as Price. The data points between the features should be same either wise it give errors.

2.2 Plotting Aggregated Values across Categories

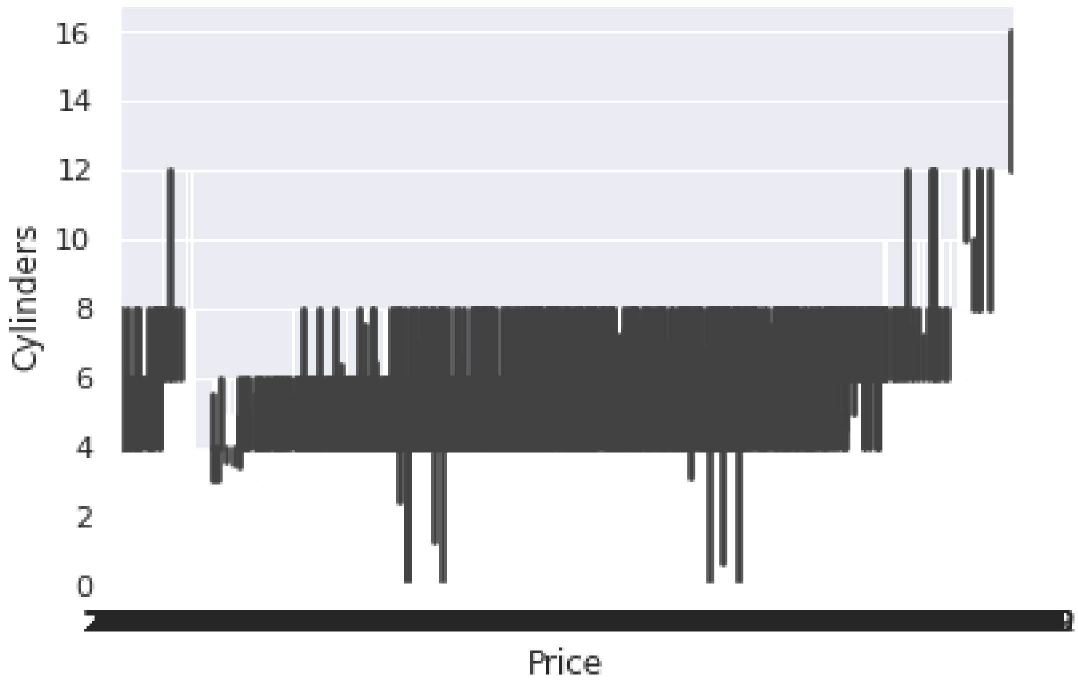
2.2.1 Bar Plots - Mean, Median and Count Plots

Bar plots are used to **display aggregated values** of a variable, rather than entire distributions. This is especially useful when you have a lot of data which is difficult to visualise in a single figure.

For example, say you want to visualise and *compare the Price across Cylinders.* The `sns.barplot()` function can be used to do that.

[98]:
```
# bar plot with default statistic=mean between Cylinder and Price

sns.barplot(x=df['Price'],y=df['Cylinders'],data=df)
# df.columns
```

[98]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f965ed3c7c0>`



2.2.2 Observation:

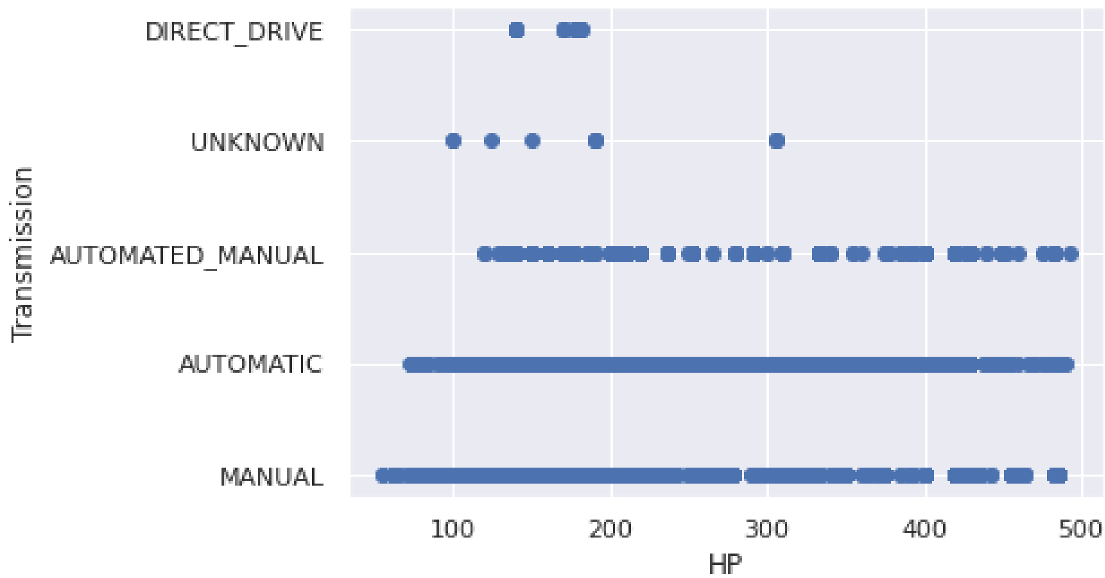
By default, seaborn plots the mean value across categories, though you can plot the count, median, sum etc. Also, barplot computes and shows the confidence interval of the mean as well.

2.3 When you want to visualise having a large number of categories, it is helpful to plot the categories across the y-axis.

2.3.1 Let's now drill down into Transmission sub categories.

[92]:
```
# Plotting categorical variable Transmission across the y-axis

plt.scatter(df2['HP'],df2['Transmission Type'])
plt.xlabel('HP')
plt.ylabel('Transmission')
# df.columns
```

[92]: `Text(0, 0.5, 'Transmission')`



These plots looks beutiful isn't it? In Data Analyst life such charts are there unavoidable friend.:)

3 Multivariate Plots

3.1 Heatmaps

A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information

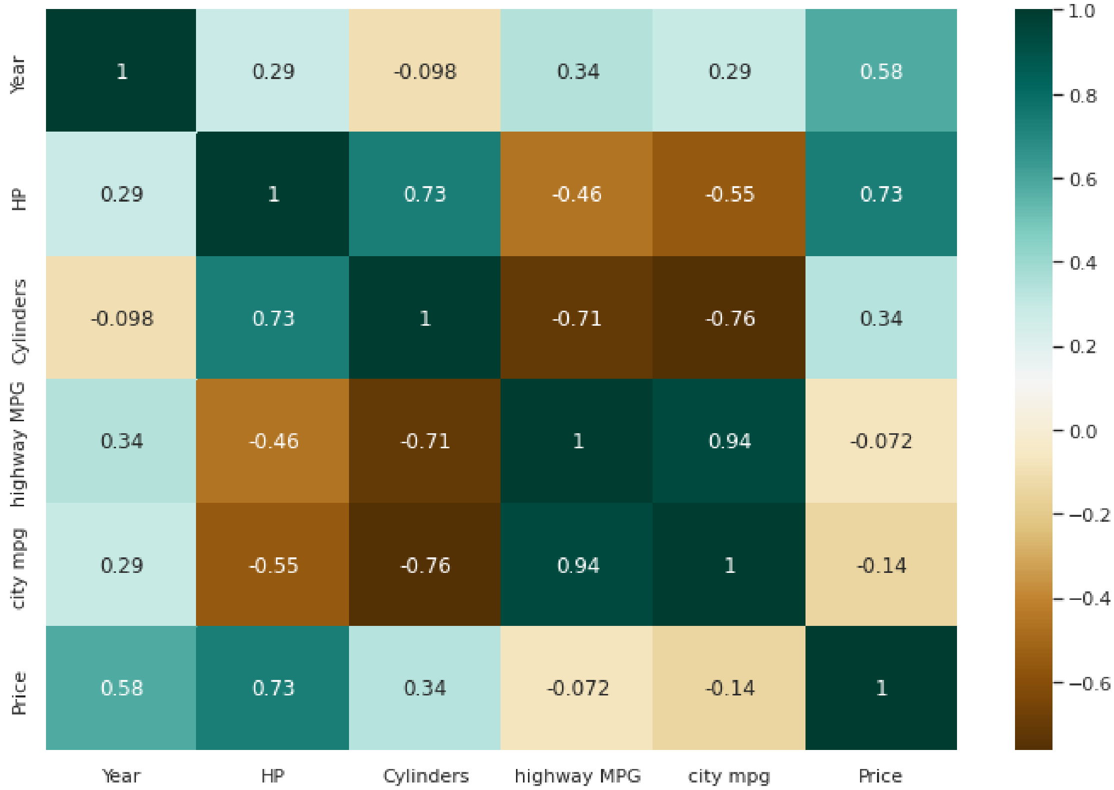
Using heatmaps plot the correlation between the features present in the dataset.

[60]:
```
#find the correlation of features of the data
corr = df2.corr()

print(corr)
```

```
                 Year        HP  Cylinders  highway MPG  city mpg     Price
Year         1.000000  0.285214  -0.098451     0.344425  0.292570  0.583586
HP           0.285214  1.000000   0.732369    -0.456375 -0.550897  0.732342
Cylinders   -0.098451  0.732369   1.000000    -0.710149 -0.763844  0.336879
highway MPG  0.344425 -0.456375  -0.710149     1.000000  0.942015 -0.072042
city mpg     0.292570 -0.550897  -0.763844     0.942015  1.000000 -0.142005
Price        0.583586  0.732342   0.336879    -0.072042 -0.142005  1.000000
```

[61]:
```
# Using the correlated df, plot the heatmap
# set cmap = 'BrBG', annot = True - to get the same graph as shown below
# set size of graph = (12,8)
plt.figure(figsize=(12,8))
```

```
sns.heatmap(corr,cmap = 'BrBG', annot = True)
plt.show()
```



3.1.1 Observation:

A heatmap contains values representing various shades of the same colour for each value to be plotted. Usually the darker shades of the chart represent higher values than the lighter shade. For a very different value a completely different colour can also be used.

The above heatmap plot shows correlation between various variables in the colored scale of -1 to 1.

[ ]:

```

```