Exploratory Data Analysis

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.

Importing the necessary libraries

In [2]:

```
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")
```

Load the dataset into dataframe

In [32]:

```
## load the csv file
df = pd.read_csv('Cars_data.csv')
```

In [33]:

```
## print the head of the dataframe

df.head()
```

Out[33]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	C
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	C

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.**

Check the datatypes

In [6]:

```
# Get the datatypes of each columns number of records in each column.
df.dtypes
```

Out[6]:

```
Make                    object
Model                   object
Year                     int64
Engine Fuel Type        object
Engine HP              float64
Engine Cylinders       float64
Transmission Type       object
Driven_Wheels           object
Number of Doors        float64
Market Category         object
Vehicle Size            object
Vehicle Style           object
highway MPG              int64
city mpg                 int64
Popularity               int64
MSRP                     int64
dtype: object
```

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

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

In [7]:

```
# initialise cols_to_drop

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

In [8]:

```
# drop the irrevalent cols and print the head of the dataframe
a = df.drop(cols_to_drop, axis=1)
# print df head
a.head()
```

Out[8]:

	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highway MPG	city mpg	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

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

In [9]:

```
# rename cols
rename_cols =["Company_name","Car_model","Year","Engine HP","Number of cylinders","Trans
mission Type","Wheel_Driven_Type","highway MPG","city mpg","Price"]
```

In [10]:

```
# use a pandas function to rename the current columns
a.columns = rename_cols
```

In [34]:

```
# Print the head of the dataframe

a.head()
```

Out[34]:

	Company_name	Car_model	Year	Engine HP	Number of cylinders	Transmission Type	Wheel_Driven_Type	highway MPG	city mpg	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

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.

In [39]:

```
# number of rows before removing duplicated rows

a[a.duplicated()]
```

Out[39]:

	Company_name	Car_model	Year	Engine HP	Number of cylinders	Transmission Type	Wheel_Driven_Type	highway MPG	city mpg	Price
14	BMW	1 Series	2013	230.0	6.0	MANUAL	rear wheel drive	28	19	31500
18	Audi	100	1992	172.0	6.0	MANUAL	front wheel drive	24	17	2000
20	Audi	100	1992	172.0	6.0	MANUAL	front wheel drive	24	17	2000
24	Audi	100	1993	172.0	6.0	MANUAL	front wheel drive	24	17	2000
25	Audi	100	1993	172.0	6.0	MANUAL	front wheel drive	24	17	2000
...	...	...	...	...	...	...	...	...	...	...
11481	Suzuki	X-90	1998	95.0	4.0	MANUAL	four wheel drive	26	22	2000
11603	Volvo	XC60	2017	302.0	4.0	AUTOMATIC	all wheel drive	29	20	46350
11604	Volvo	XC60	2017	240.0	4.0	AUTOMATIC	front wheel drive	30	23	40950
11708	Suzuki	XL7	2008	252.0	6.0	AUTOMATIC	all wheel drive	22	15	29149
11717	Suzuki	XL7	2008	252.0	6.0	AUTOMATIC	front wheel drive	22	16	27499

989 rows × 10 columns

In [40]:

```
# drop the duplicated rows
df = a.drop_duplicates()

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

Out[40]:

	Company_name	Car_model	Year	Engine HP	Number of cylinders	Transmission Type	Wheel_Driven_Type	highway MPG	city mpg	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

In [41]:

```
# Count Number of rows after deleting duplicated rows

df.shape
```

Out[41]:

Out[41]:

```
(10925, 10)
```

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

In [15]:

```
# check for nan values in each columns
df.isnull().sum()
```

Out[15]:

```
Company_name           0
Car_model              0
Year                   0
Engine HP             69
Number of cylinders   30
Transmission Type      0
Wheel_Driven_Type      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.

In [16]:

```
# drop missing values
df = df.dropna()
```

In [17]:

```
# Make sure that missing values are removed
# check number of nan values in each col again
df.isnull().sum()
```

Out[17]:

```
Company_name           0
Car_model              0
Year                   0
Engine HP              0
Number of cylinders    0
Transmission Type      0
Wheel_Driven_Type      0
highway MPG            0
city mpg               0
Price                  0
dtype: int64
```

In [18]:

```
#Describe statistics of df
df.describe()
```

Out[18]:

Out[18]:

	Year	Engine HP	Number of cylinders	highway MPG	city mpg	Price
count	10827.000000	10827.000000	10827.000000	10827.000000	10827.000000	1.082700e+04
mean	2010.896370	254.553062	5.691604	26.308119	19.327607	4.249325e+04
std	7.029534	109.841537	1.768551	7.504652	6.643567	6.229451e+04
min	1990.000000	55.000000	0.000000	12.000000	7.000000	2.000000e+03
25%	2007.000000	173.000000	4.000000	22.000000	16.000000	2.197250e+04
50%	2015.000000	240.000000	6.000000	25.000000	18.000000	3.084500e+04
75%	2016.000000	303.000000	6.000000	30.000000	22.000000	4.330000e+04
max	2017.000000	1001.000000	16.000000	354.000000	137.000000	2.065902e+06

Removing outliers

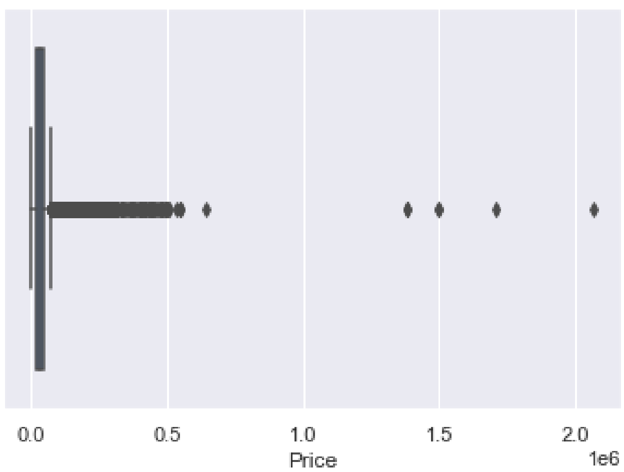
Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called outliers and often machine learning modeling and model skill in general can be improved by understanding and even removing these outlier values.

In [19]:

```
## Plot a boxplot for 'Price' column in dataset.

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

Out[19]:

<AxesSubplot:xlabel='Price'>



Observation:

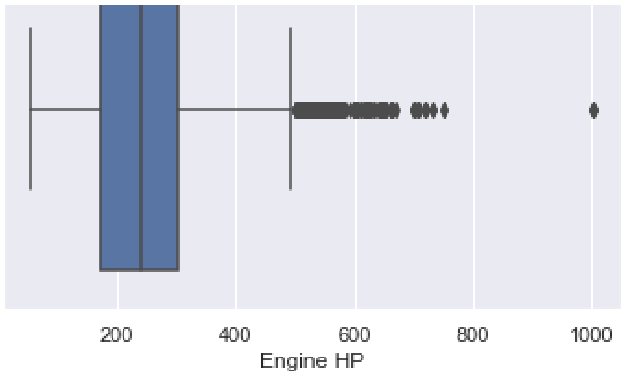
Here as you see that we got some values near to 1.5 and 2.0 . So these values are called outliers. Because there are away from the normal values. Now we have detect the outliers of the feature of Price. Similarly we will checking of anothers features.

In [20]:

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

Out[20]:

<AxesSubplot:xlabel='Engine HP'>




```
Observation:
```

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

In [ ]:

```

```

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

Hint: Use loc with condition

In [21]:

```
# print all the columns which are of int or float datatype in df.
df = df.select_dtypes(include=['float64','int64'])
df
```

Out[21]:

	Year	Engine HP	Number of cylinders	highway MPG	city mpg	Price
0	2011	335.0	6.0	26	19	46135
1	2011	300.0	6.0	28	19	40650
2	2011	300.0	6.0	28	20	36350
3	2011	230.0	6.0	28	18	29450
4	2011	230.0	6.0	28	18	34500
...	...	...	...	...	...	...
11909	2012	300.0	6.0	23	16	46120
11910	2012	300.0	6.0	23	16	56670
11911	2012	300.0	6.0	23	16	50620
11912	2013	300.0	6.0	23	16	50920
11913	2006	221.0	6.0	26	17	28995

10827 rows × 6 columns

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

In [22]:

```
# save column names of the above output in variable list
l=df.select_dtypes(include=['float64','int64']).columns
l
```

Out[22]:

```
Index(['Year', 'Engine HP', 'Number of cylinders', 'highway MPG', 'city mpg',
```

```
       'Price'],
      dtype='object')
```

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)

In [23]:

```python
## define Q1 and Q2
Q1 = np.percentile(df['Engine HP'],25)
Q3 = np.percentile(df['Engine HP'],75)

# # define IQR (interquantile range)
IQR = Q3 - Q1
lower_limit = (Q1-1.5*IQR)
upper_limit = (Q3+1.5*IQR)

# # define df2 after removing outliers
df2 = df[(df['Engine HP']>lower_limit) & (df['Engine HP'] < upper_limit)]
```

In [24]:

```python
# find the shape of df & df2

print(df.shape)
print(df2.shape)
```

```
(10827, 6)
(10332, 6)
```

In [25]:

```python
# find unique values and there counts in each column in df using value counts function.

for i in df.columns:
    b=str(i)
    unique = pd.unique(df[b])
    print('Number of unique value in df[{}] is ' .format(b),len(unique),'and they are',u
nique)
```

```
Number of unique value in df[Year] is  28 and they are [2011 2012 2013 1992 1993 1994 201
7 1991 2016 1990 2015 1996 1997 1998
 2014 1999 2002 2003 2004 1995 2007 2008 2009 2001 2010 2000 2005 2006]
Number of unique value in df[Engine HP] is  355 and they are [ 335.  300.  230.  320.  17
2.  160.  130.  158.  240.  248.  162.  217.
  184.  295.  115.  140.  155.  114.  100.  241.  180.  177.  228.  121.
  148.  194.  218.  161.  292.  250.  255.  222.   82.  134.  306.  400.
  425.  350.  332.  268.  282.  275.  201.  442.  562.  597.  237.  270.
  445.  443.  302.  322.  315.  101.  135.  485.  238.  515.  543.  631.
  604.  620.  611.  661.  157.  402.  389.  110.  532.  170.  165.  125.
  641.  535.  153.  144.  188.  372.  108.  168.  190.  205.  200.  227.
  173.  220.  210.  280.  207.  265.  260.  290.  285.  390.  225.  185.
  150.  430.  520.  560.  475.  500.  540.  370.  580.  420.  345.  195.
  193.  208.  181.  236.  186.  252.  310.  333.  340.  450.  281.  288.
  138.  137.  106.  271.  196.  212.  278.  189.  480.  152.  600.  375.
  198.  182.  179.  264.  503.  456.  317.  235.  385.  303.   63.  321.
  272.  464.  202.  215.  283.  700.  720.  750.  107.  293.  119.  143.
  245.  120.  337.  276.  330.  132.  199.  530.  451.  329.  469.  362.
   94.  553.  453.  483.  323.  426.  505.  455.  650.  178.  242.  305.
```

```
 605.  440.  570.  325.  175.  707.  131.   62.   92.  102.  127.  174.
 621.  510.  429.  536.  355.  382.  577.  113.  136.  234.  552.  626.
 616.  572.  521.  567.  582.  460.  164.  192.  224.  239.  404.  318.
 556.  640.  122.  146.  244.  273.  563.  141.  435.  550.  360.  145.
 349.  166.  147.  128.  197.  291.  660.  261.  156.  403.   95.  297.
  81.  257.  365.  203.  231.  731.  651.  287.  123.  126.  416.  343.
 348.  328.  298.  171.  219.  221.  311.  361.  256.  415.  274.  449.
 395.  401.  454.  444.  338.  342.  467.  545.  565.  301.  263.   93.
 187.  610.  111.   98.  204.  211.   73.   66.  304.  381.  142.   74.
 424.  253.   90.  386.  359.  438.  232.  383.  518.  493.  259.  523.
  55.   79.  116.  151.   78.  191.  592.  632.  670.   88.  167.  118.
 380.  214.  573.  284.   99.  103.  525.  254.  470.  176.  279.  377.
 251.  223.  308.  105.  316.  124.  526.  662.  266.  296.  557.  617.
 583.  622.   84.  163.  354.  159.   96.  206.  169.  133.  568.  109.
1001.  645.  490.  624.  410.   97.  394.]
Number of unique value in df[Number of cylinders] is  9 and they are [ 6.   4.   5.   8. 12.
0. 10.  3. 16.]
Number of unique value in df[highway MPG] is  44 and they are [ 26  28  27  25  24  20  2
1  22  35  34  31  30  32  33  23  36  29  45
  43  40  42  19  18  17  15  37  39  41  16  14  38  12 354  47  46  82
  44  13 111 106  48  53  50 109]
Number of unique value in df[city mpg] is  50 and they are [ 19  20  18  17  16  26  23
22  21  24  15  25  29  28  32  31  30  14
  10  27  12  13   9  11   8  50  49  47  35  33  40  85  42  43  36  44
   7  34 137 129  39  41  37  53  55  51  54  58  38 128]
Number of unique value in df[Price] is  6014 and they are [46135 40650 36350 ... 46120 50
620 50920]
```

Visualising Univariate Distributions

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

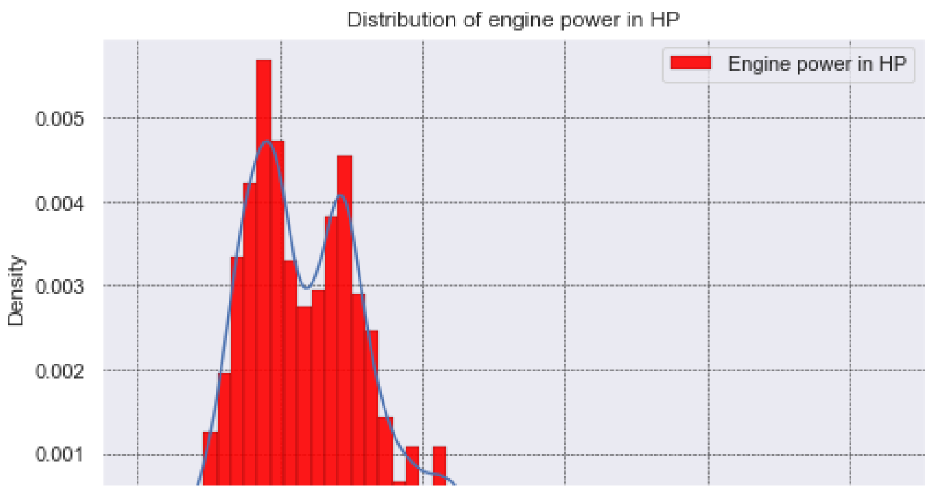
Do you know? you have just now already explored one univariate plot. guess which one? Yeah its box plot.

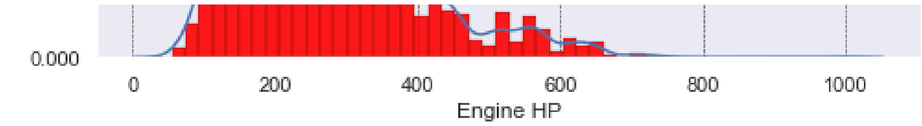
Histogram & Density Plots

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`.

In [26]:

```python
#ploting distplot for variable HP

plt.figure(figsize=(8,5))
sns.distplot(df['Engine HP'],kde=True,hist_kws={'color':'red','edgecolor':'black','alpha
':0.9,'linewidth':0.2},label = 'Engine power in HP')
plt.title('Distribution of engine power in HP')
plt.grid(color='k',linestyle='--',linewidth=0.5)
plt.legend()
plt.show()
```





`Observation:`

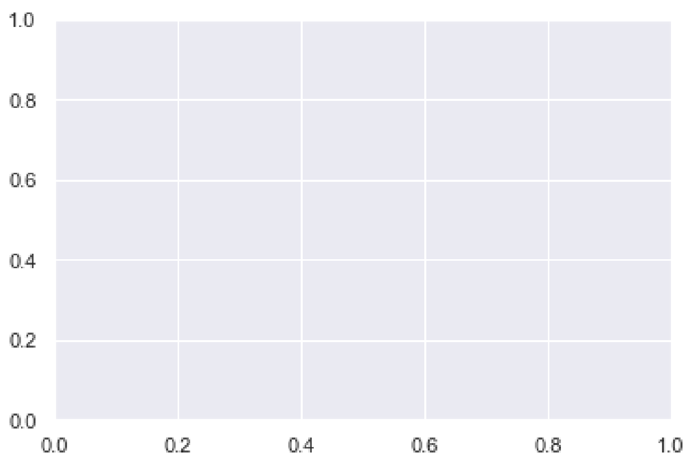
We plot the Histogram of feature HP with help of distplot in seaborn.
In this graph we can see that there is max values near at 200. similary we have also the 2nd highest value near 400 and so on.
It represents the overall distribution of continuous data variables.

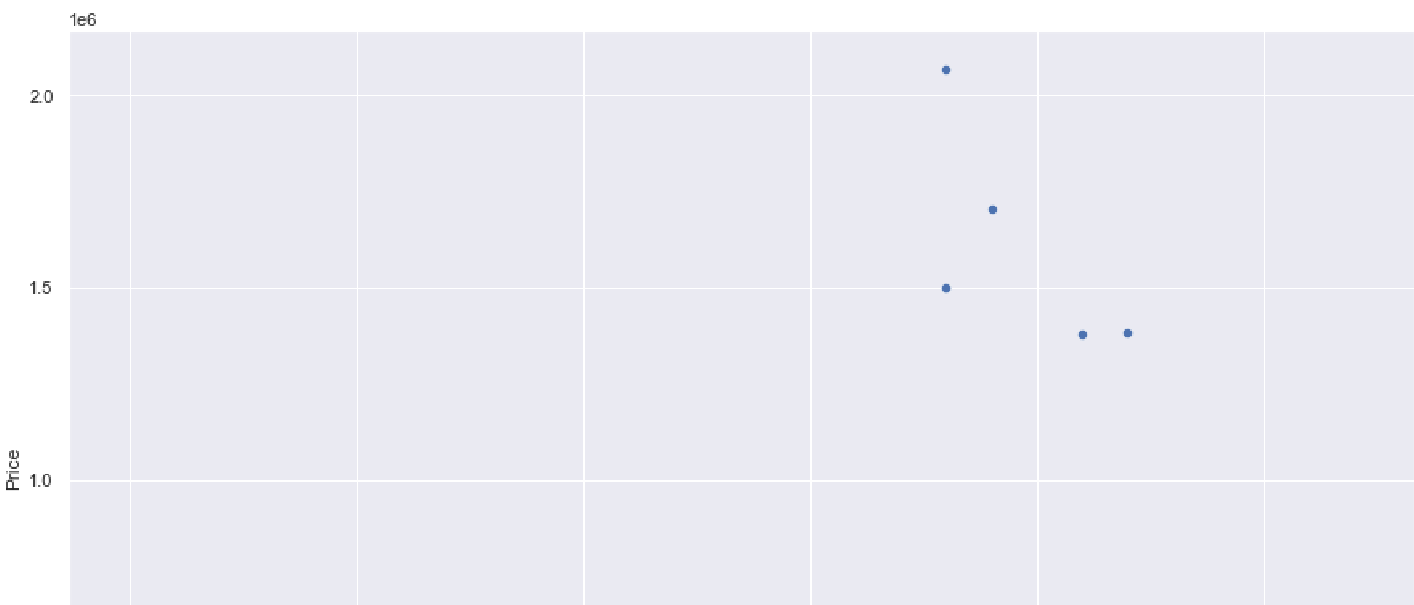
Since seaborn uses matplotlib behind the scenes, the usual matplotlib functions work well with seaborn. For example, you can use subplots to plot multiple univariate distributions.

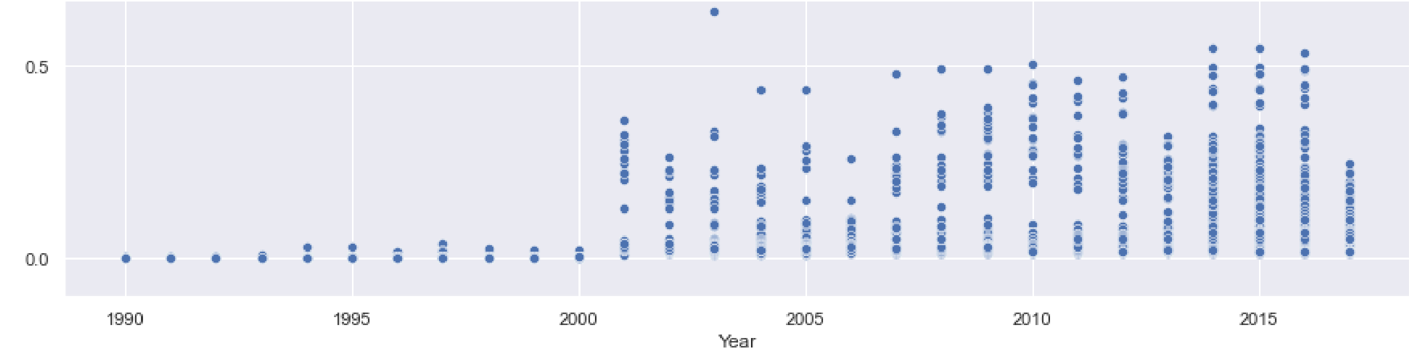
- **Hint: use matplotlib subplot function**

In [27]:

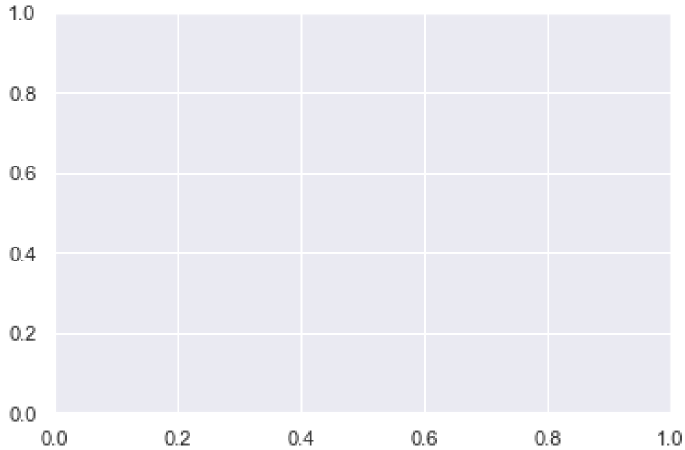
```
# plot all the columns present in list l together using subplot of dimention (2,3).


c=1

for i in l:
    plt.subplot(1,1,c)
    plt.figure(figsize=(15,10))
    sns.scatterplot(data=df,x=df[i],y=df['Price'])
    plt.show()
    c=+1

# plt.show()
```



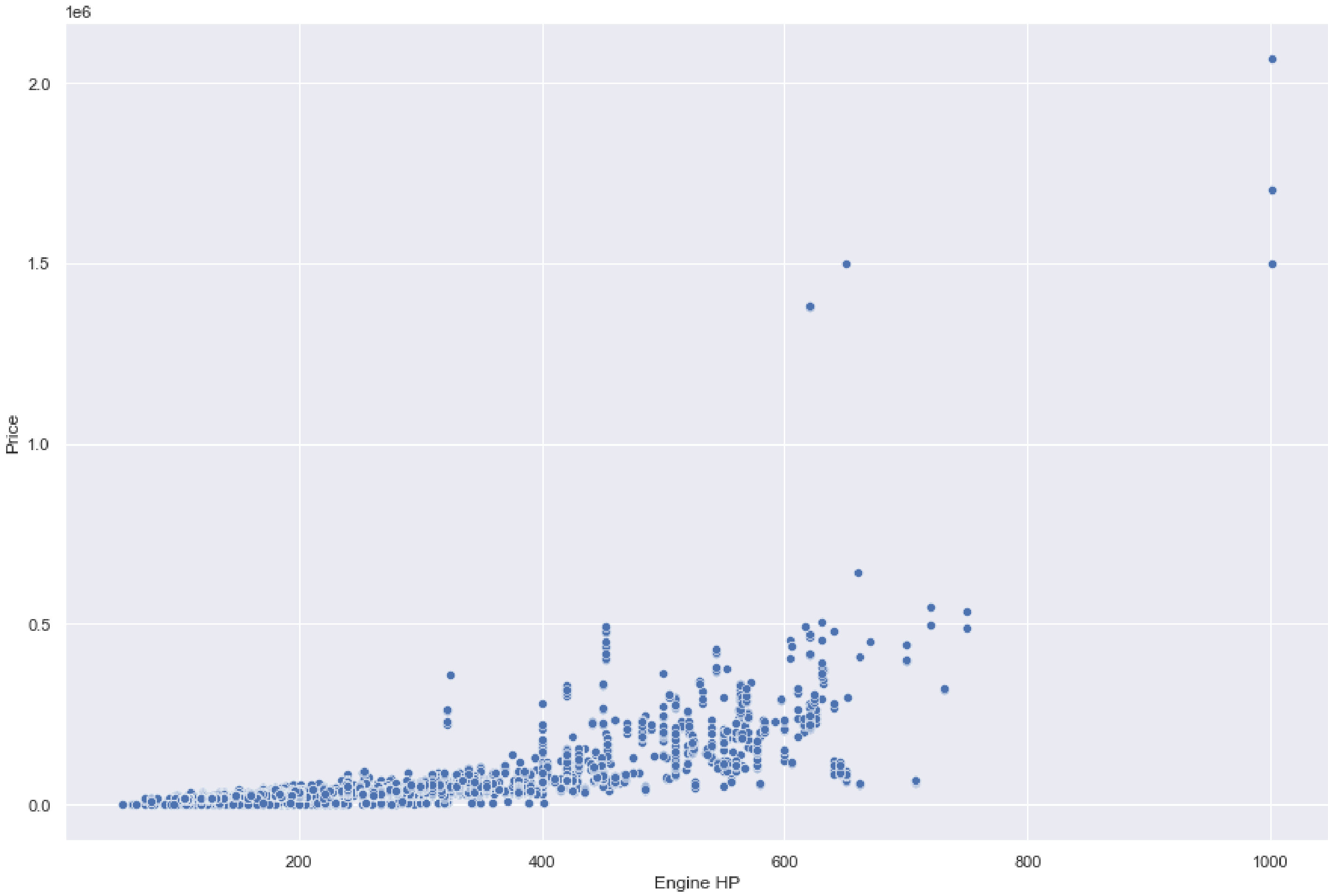




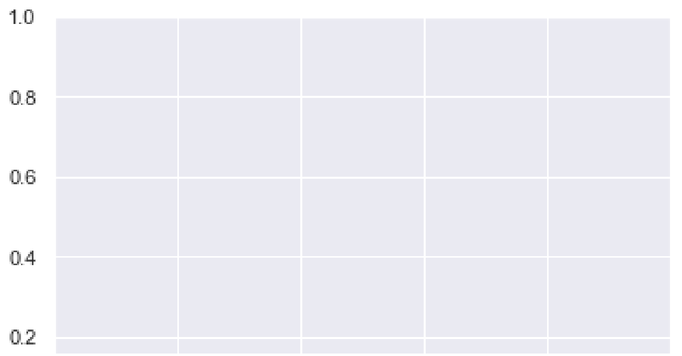
0.5
0.0
1990
1995
2000
2005
2010
2015
Year



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

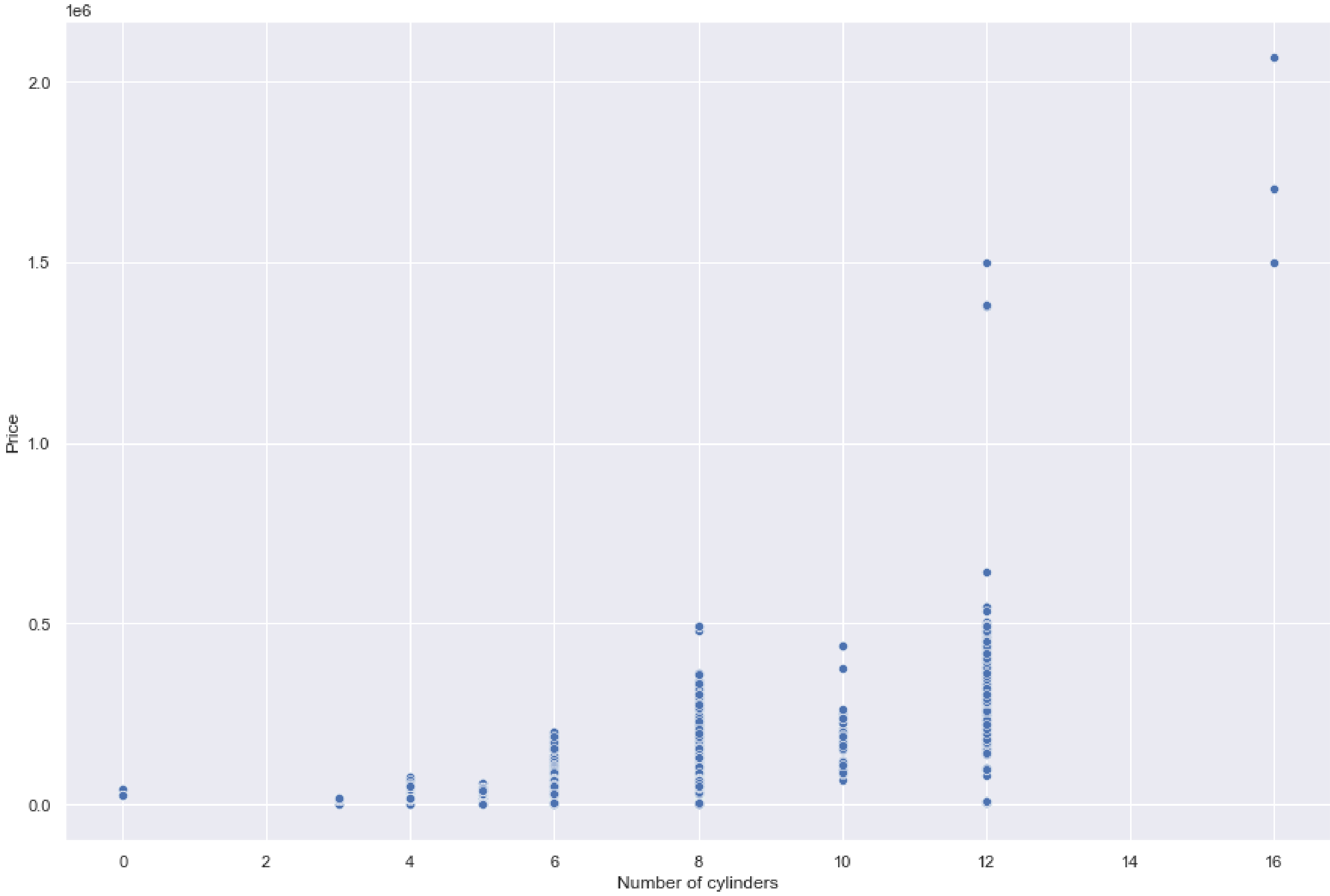


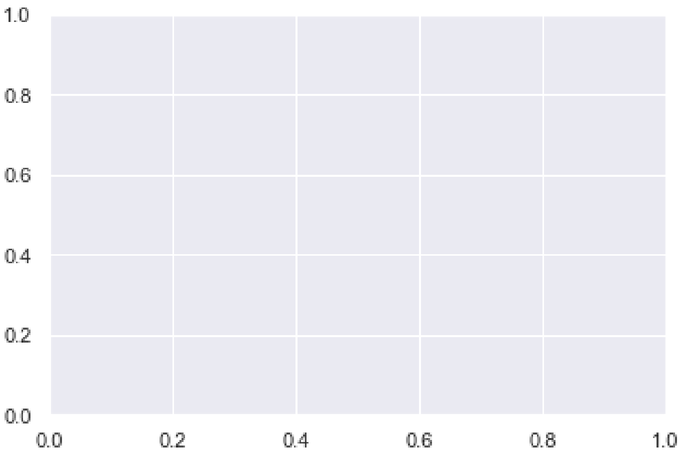
1e6
2.0
1.5
1.0
0.5
0.0
Price
200
400
600
800
1000
Engine HP

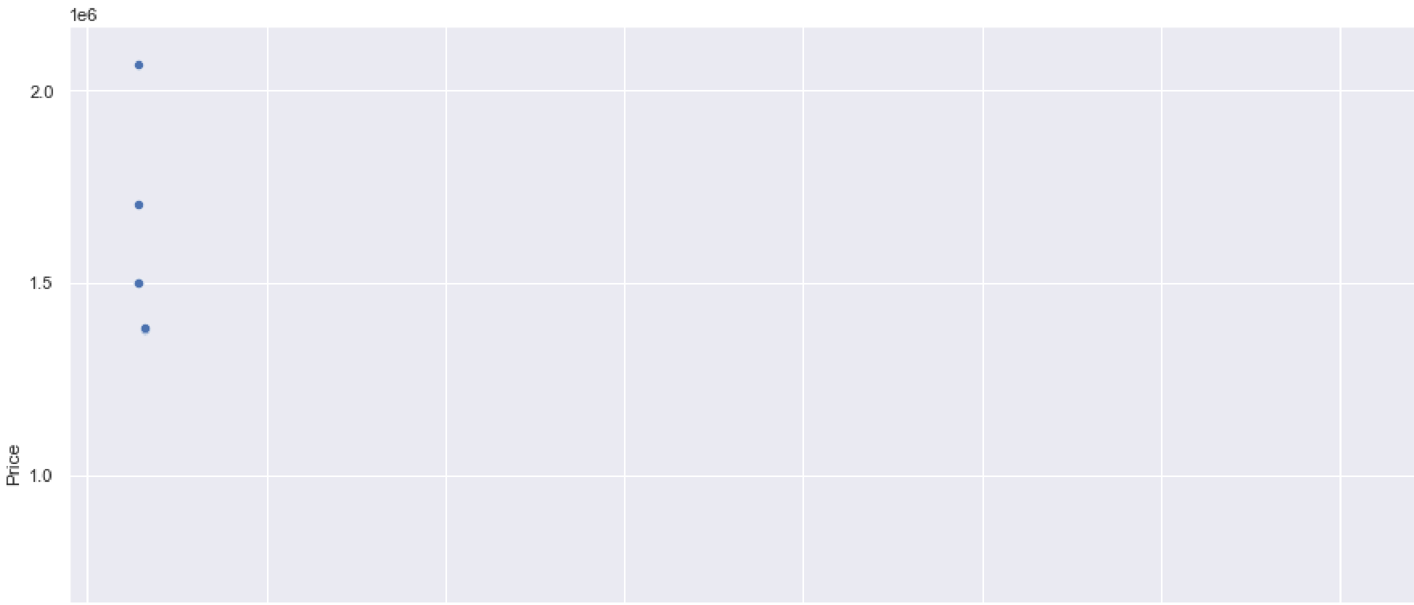


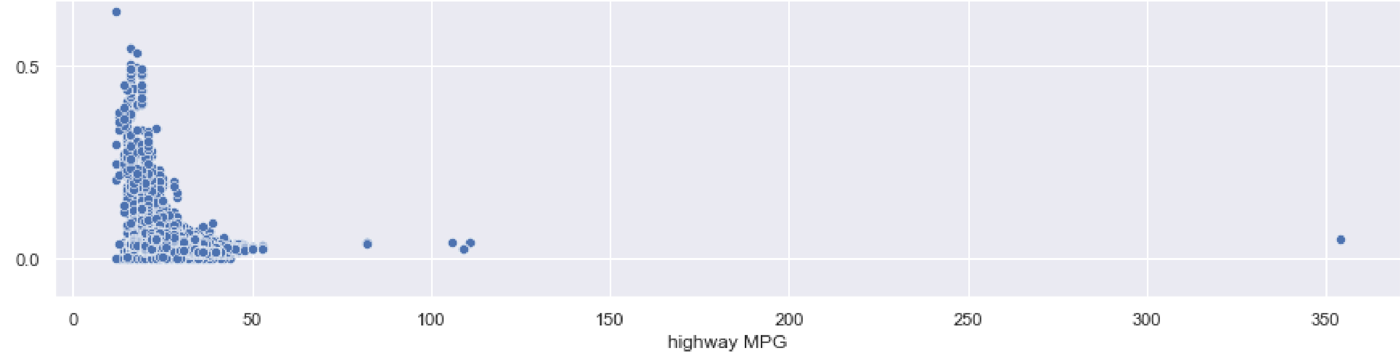
1.0
0.8
0.6
0.4
0.2

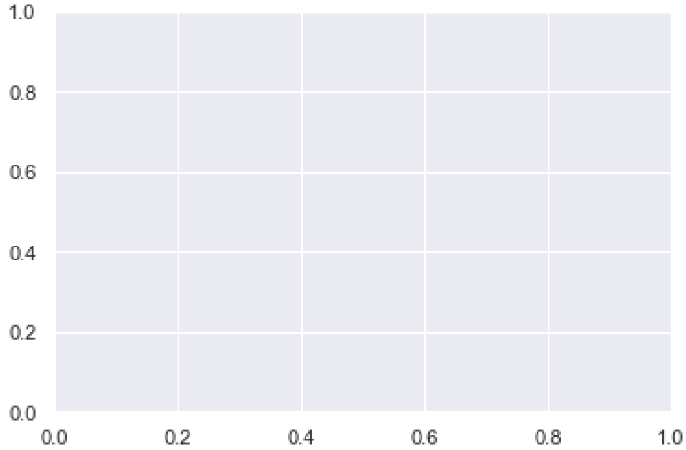

0.0
0.0 0.2 0.4 0.6 0.8 1.0

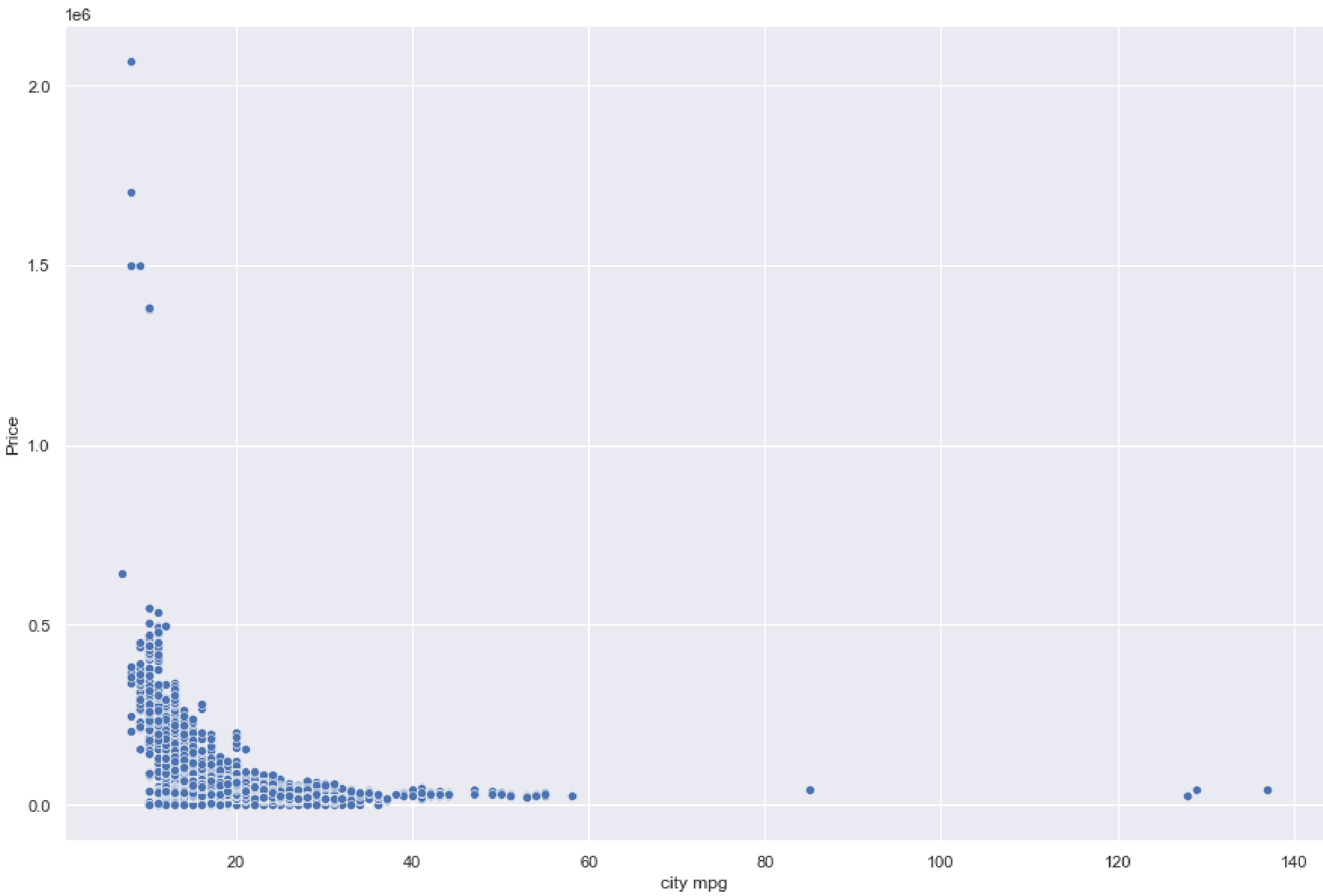

1e6
2.0
1.5
Price 1.0
0.5
0.0
0 2 4 6 8 10 12 14 16
Number of cylinders

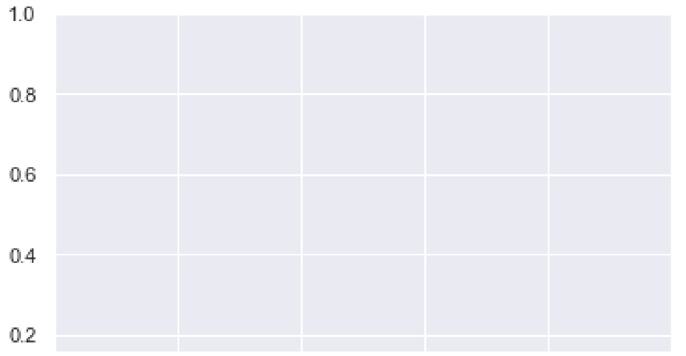

1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0

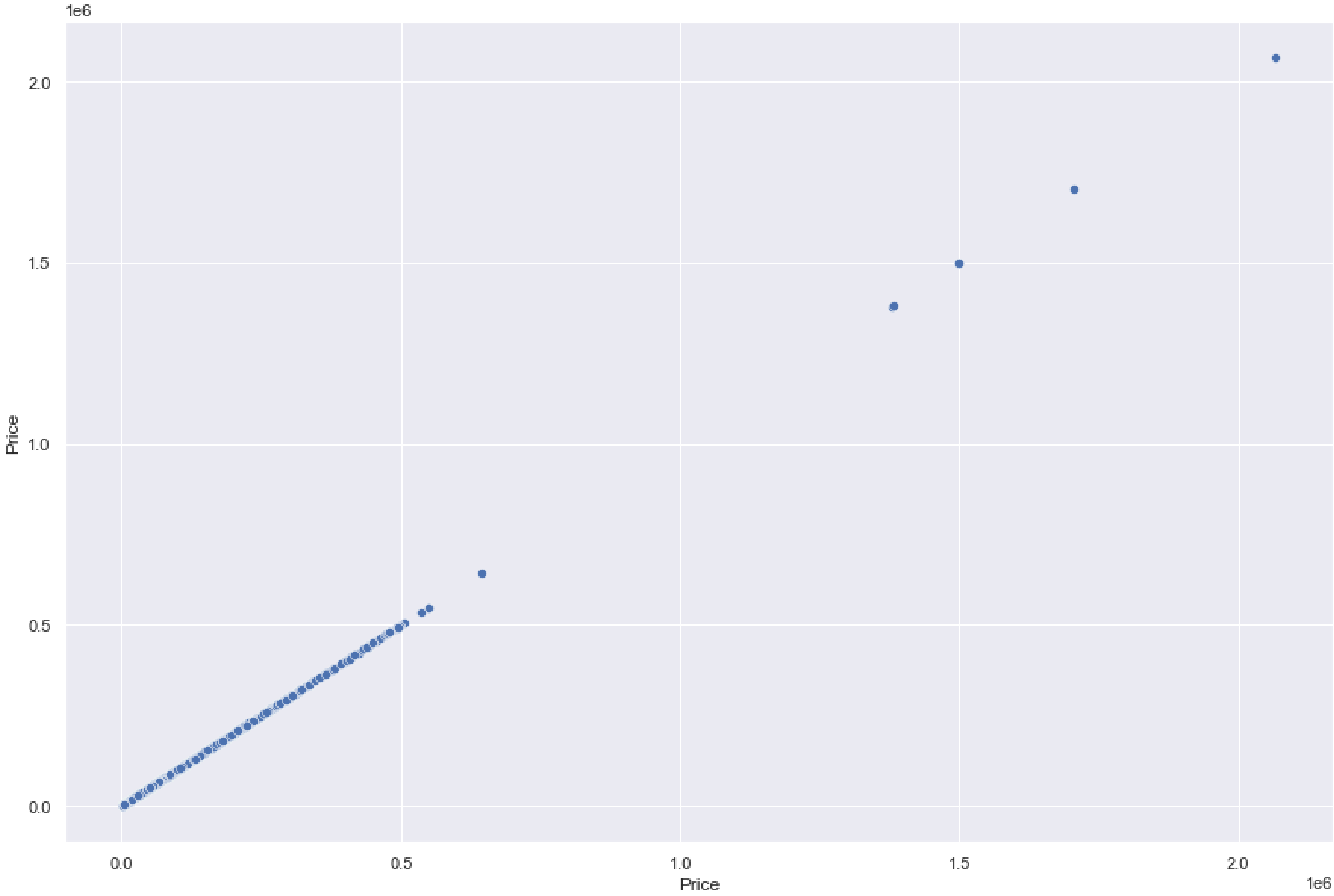

1e6
2.0
1.5
Price 1.0


0.5
0.0
0
50
100
150
200
250
300
350
highway MPG


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1e6
2.0
1.5
1.0
0.5
0.0
Price
20
40
60
80
100
120
140
city mpg


1.0
0.8
0.6
0.4
0.2

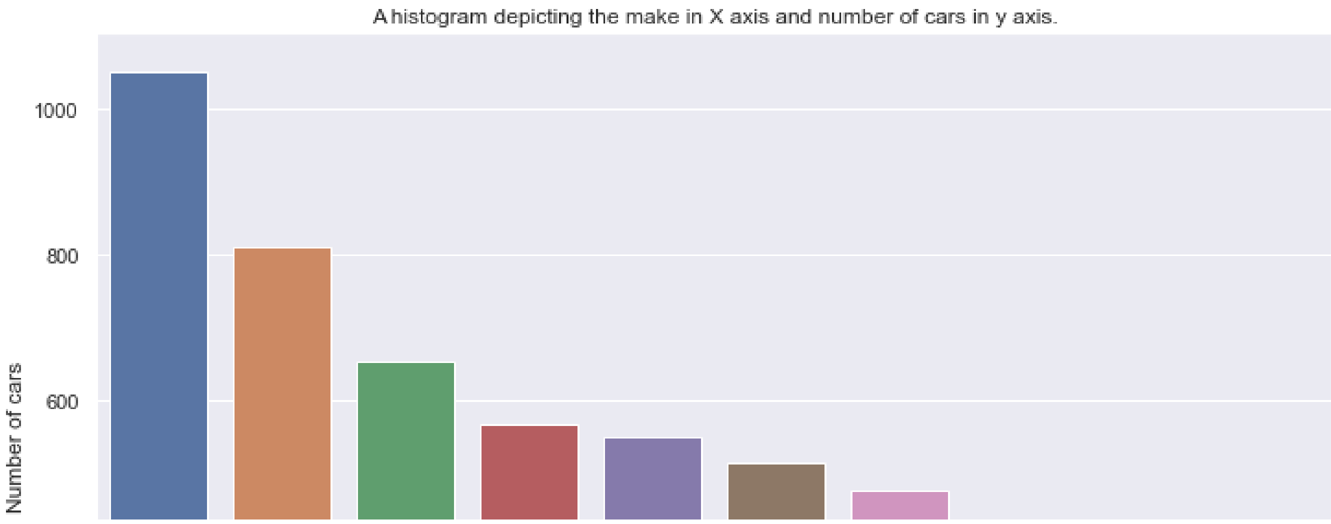


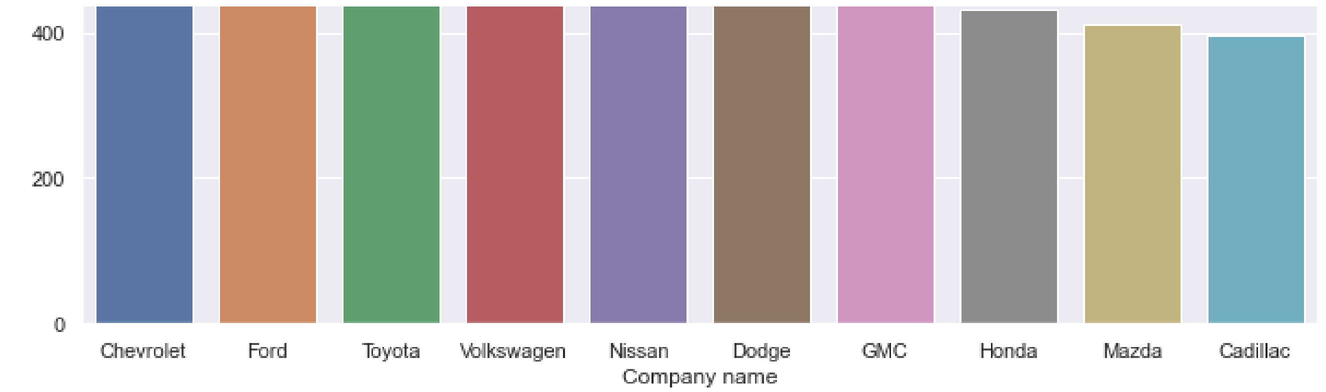
Bar Chart Plots

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

In [42]:

```
plt.figure(figsize = (12,8))
sns.countplot(data=df,x='Company_name',order=pd.value_counts(df['Company_name']).iloc[:1
0].index)
# use nlargest and then .plot to get bar plot like below output
plt.title('A histogram depicting the make in X axis and number of cars in y axis.')
# Plot Title, X & Y label
plt.xlabel('Company name')
plt.ylabel('Number of cars')
plt.show()
```





`Observation:`

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

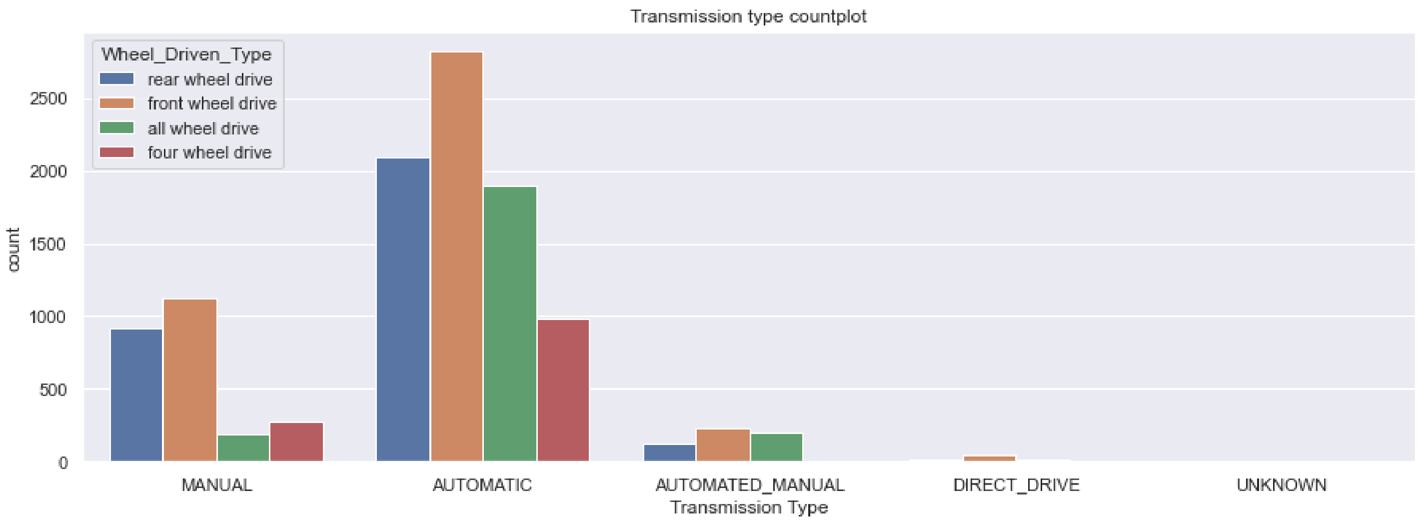
`Count Plot`

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

In [43]:

```
plt.figure(figsize=(15,5))
sns.countplot(data=df,x='Transmission Type',hue='Wheel_Driven_Type')
plt.title('Transmission type countplot')

# plot countplot on transmission and drive mode

plt.xlabel('Transmission Type')
plt.ylabel('count')
plt.show()
```



`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.

`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.

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.

In [45]:

```
## Your code here -
plt.figure(figsize=(12,8))
fig, ax = plt.subplots(figsize=(14,10))

# plot scatterplot on hp and price
sns.scatterplot(data=df,x='Engine HP',y='Price',hue='Transmission Type',style='Transmiss
ion Type',size='Transmission Type')
plt.show()
```

```
<Figure size 864x576 with 0 Axes>
```

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.

Plotting Aggregated Values across Categories

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.

In [46]:

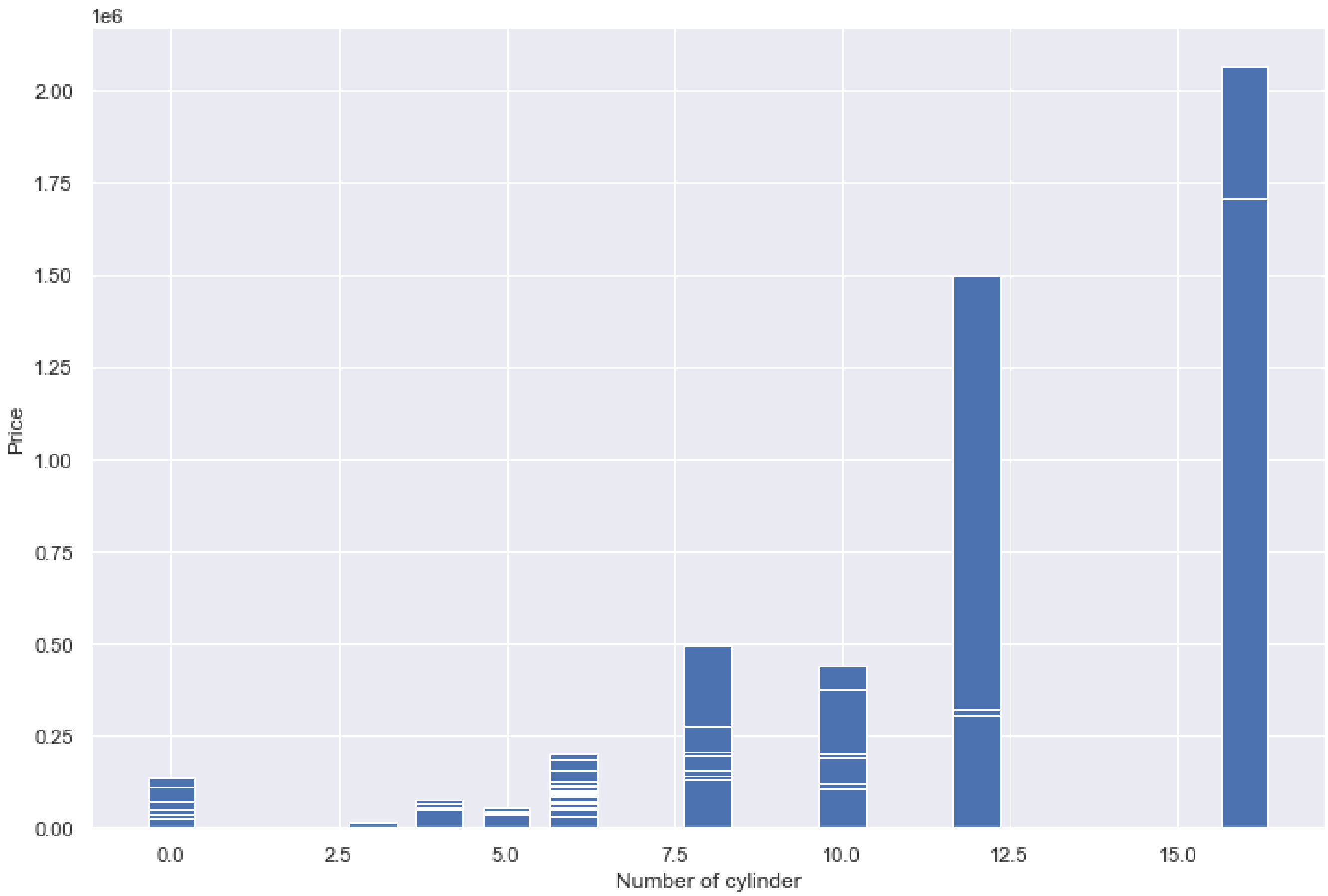
```
df.head()
```

Out[46]:

	Company_name	Car_model	Year	Engine HP	Number of cylinders	Transmission Type	Wheel_Driven_Type	highway MPG	city mpg	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

In [55]:

```
# bar plot with default statistic=mean between Cylinder and Price

plt.figure(figsize=(12,8))
plt.bar(x=df['Number of cylinders'],height=df['Price'],width=0.7,color='b')
plt.xlabel('Number of cylinder')
plt.ylabel('Price')
plt.show()
```



`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.

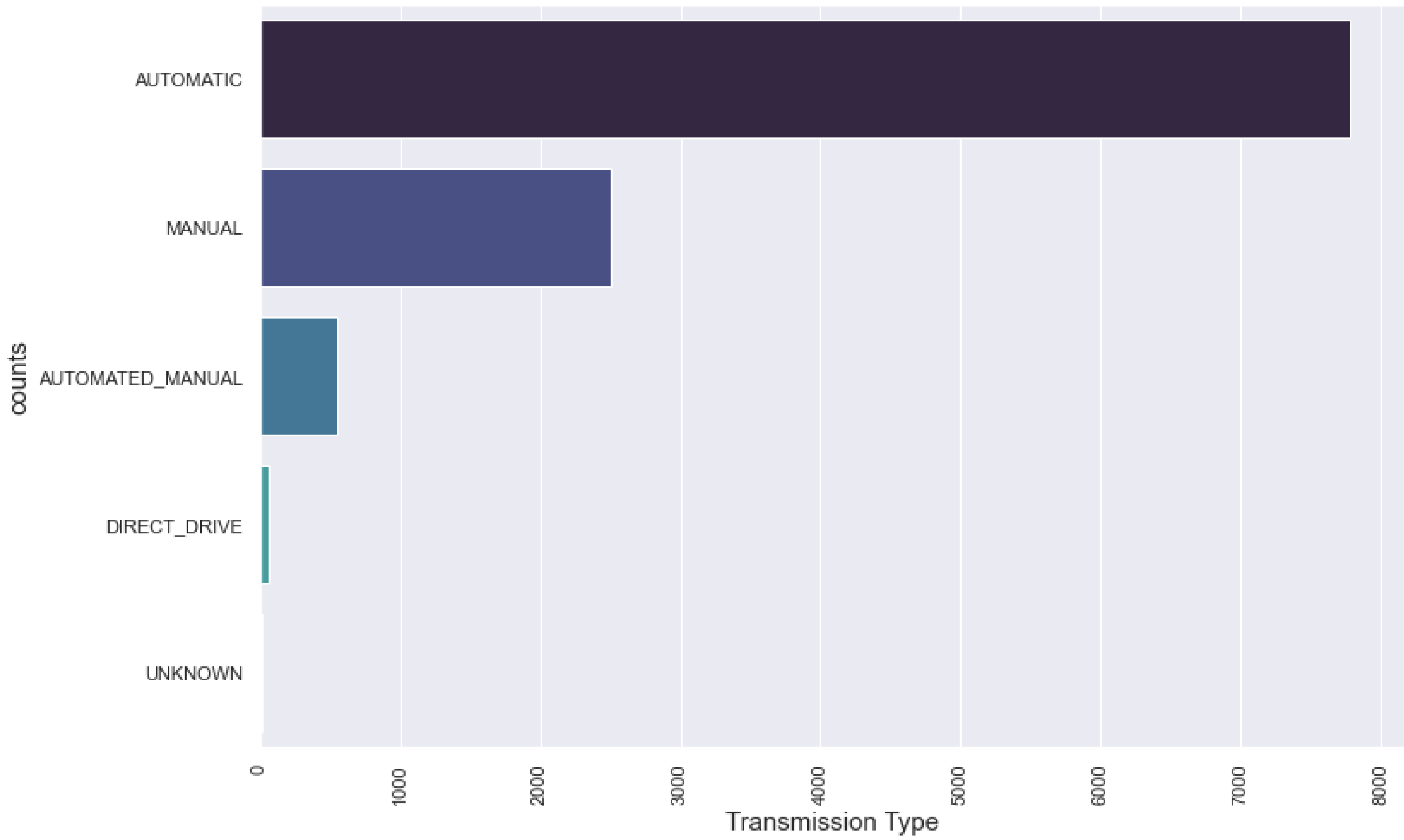
`When you want to visualise having a large number of`

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

Let's now drill down into Transmission sub categories.

In [57]:

```
# Plotting categorical variable Transmission across the y-axis

plt.figure(figsize=(12,8))
sns.barplot(df['Transmission Type'].value_counts().values ,df['Transmission Type'].value
_counts().index, palette='mako')
plt.xticks(rotation=90)
plt.xlabel("Transmission Type", fontsize=15)
plt.ylabel('counts',  fontsize=15)
plt.show( )
```



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

Multivariate Plots

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.

In [58]:

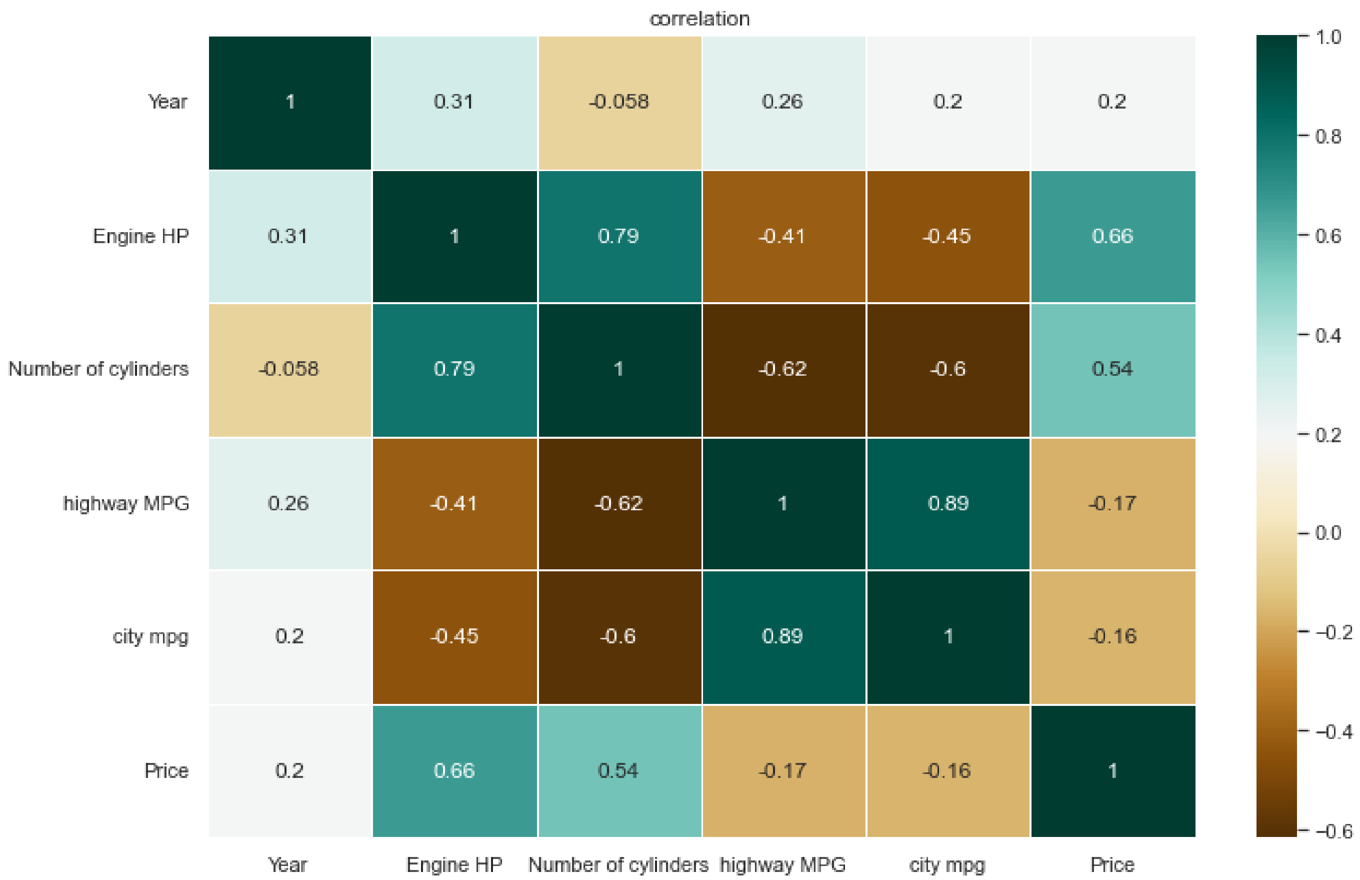
```
#find the correlation of features of the data
corr = df.corr()

# print corr
corr
```

Out[58]:

	Year	Engine HP	Number of cylinders	highway MPG	city mpg	Price
Year	1.000000	0.313833	-0.057691	0.259907	0.198013	0.197071
Engine HP	0.313833	1.000000	0.788007	-0.412052	-0.445661	0.659568
Number of cylinders	-0.057691	0.788007	1.000000	-0.615148	-0.597641	0.540688
highway MPG	0.259907	-0.412052	-0.615148	1.000000	0.885991	-0.167339
city mpg	0.198013	-0.445661	-0.597641	0.885991	1.000000	-0.163052
Price	0.197071	0.659568	0.540688	-0.167339	-0.163052	1.000000

In [59]:

```python
# 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(df.corr(),annot=True,cmap='BrBG',linewidth=1)
plt.title('correlation')
plt.show()
```



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.

In [ ]: