A Report on:

Exploratory data analysis & Linear Regression on Automobile Dataset

Estimation of Automobile Price

Done By:

Sai Lokesh Reddy

Computer science engineering

(2018-2022)

Lovely professional university Punjab

A report submitted in the fulfilment of the requirements for the Internship certification of the course 'DATA ANALYTICS & MACHINE LEARNING'.

&

Submitted to:

Mr. Mayur Dev Sewak
General Manager, Operations
EISystems Services.

&

Mr. Prateek Misra
Trainer, ICT Domain, Techsimplus
EISystems Services.

Table of Contents

Title of the Project: Exploratory Data Analysis & Linear Regression on Automobile Dataset

Date of Submission: August 8, 2020

Team Members:

1. Rithika Chavan
2. Lokesh Reddy

ABSTRACT & PROJECT SUMMARY:

Machine learning is broadly classified into 3 types: Supervised Learning, Unsupervised learning and Reinforcement learning. In this report we shall be focusing on one of the algorithms based on **Supervised Learning** which is ***Linear Regression*** and also perform **Exploratory Data Analysis (*EDA*)** on an **Automobile Dataset.**

The dataset is from 1985 Ward's Automotive Yearbook. This data set consists of three types of entities:

- The specification of an automobile in terms of various characteristics
- The assigned insurance risk rating
- The normalized losses in use as compared to other cars

Supervised learning: In this, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.
EDA is a practice of iteratively asking a series of questions about the data at your hand and trying to build hypotheses based on the insights you gain from the data.
At an EDA phase, one of the algorithms we often use is **Linear Regression**.
Linear Regression is an algorithm to draw an optimized straight line between two or more variables. Being able to draw such a straight line helps us not only predict the unknown but also understand the relationship between the variables better. Though it has been there for a long time, it is still the most often used algorithm among many data scientists thanks to its simplicity.

With the recent popularity of Machine Learning algorithms, there has been a lot of attention focused on the prediction side of things. But what we find the most useful for this type of Statistical algorithm is its ability to help us investigate the relationship between the variables.

Through our report, we also demonstrate how Linear Regression is useful to produce useful insights on the given automobile dataset in estimating automobile prices and help us build good hypotheses effectively at Exploratory Data Analysis (EDA) phase.

OBJECTIVES OF THE PROJECT:

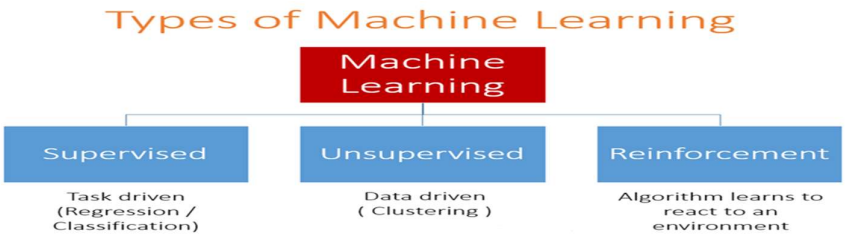
i. INTRODUCTION:

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Machine learning algorithms are often categorized as supervised or unsupervised:

- **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training.
- **Unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.
- **Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy.
- **Reinforcement machine learning algorithms** is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance.

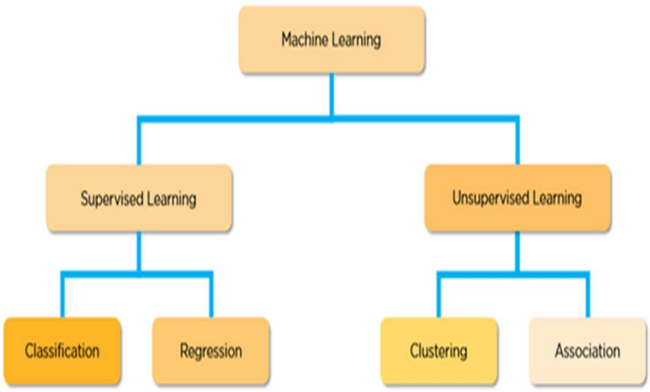


Exploratory Data Analysis:

In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

EDA is a practice of iteratively asking a series of questions about the data at your hand and trying to build hypotheses based on the insights you gain from the data.

ii. Motivation, Plan of the Project, Methodologies:



In this report few methods for **analyzing**, **cleaning**, **data visualization** and **introduction to linear regression** have been explained. For Linear regression I used a *sklearn* and *stats model comparison* in evaluating the model, to check the commensuration. Furthermore as an important part of the linear regression, the calculation of **Regression Evaluation Metrics** has also been included.

Here is the dataset from 1985 Ward's Automotive Yearbook. This data set consists of three types of entities:

1. The specification of an auto in terms of various characteristics
2. The assigned insurance risk rating
3. The normalized losses in use as compared to other cars

Moreover there is a risk factor assigned for the cars, based on its price.

<u>The Methodology</u>:
The complete steps in *estimating the price of the cars* (our goal) is achieved through the following steps:

- **Getting the environment ready with required libraries and importing the data.**
- **Checking out the Data** - Looking around the Dataset and its information.
- **Missing Data** - Handling the missing data using different techniques.
- **Exploratory Data Analysis (EDA)** - Exploring the different features and its behavior.
- **Correlation between different features** - Using Heat maps in studying correlation.
- **Data cleaning** - For model development; irrelevant, incomplete, incorrect or inaccurate variables are Identified incomplete and these data are replaced, modifying, or deleted.
- **Building Regression Model** - The dataset is split for training and testing.
- **Predictions from Model** - This is to look at how well the model behaved in predicting the outcome.
- **Calculating the P-Value** - This is to ensure whether the data really represent the observed effect.
- **Regression Evaluation Metrics** - These are the common evaluation metrics for regression problems.

INPUT DATASET:

This dataset consist of data From 1985 Ward's Automotive Yearbook. Here are the sources:

1) 1985 Model Import Car and Truck Specifications, 1985 Ward's Automotive Yearbook.
2) Personal Auto Manuals, Insurance Services Office, 160 Water Street, New York, NY 10038
3) Insurance Collision Report, Insurance Institute for Highway Safety, Watergate 600, Washington, DC 20037

This data set consists of three types of entities:

(a) The specification of an auto in terms of various characteristics,

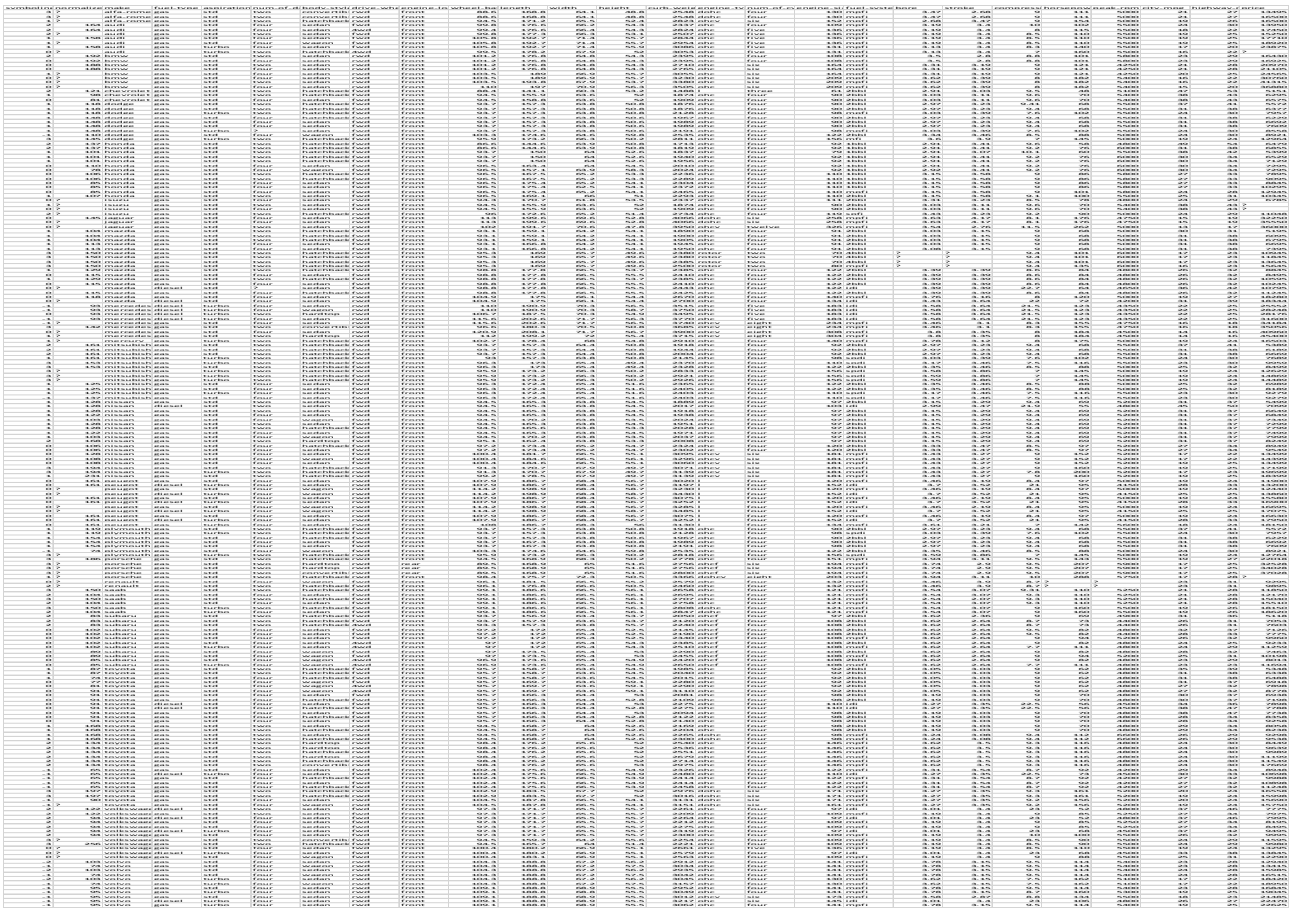
(b) Its assigned insurance risk rating,

(c) Its normalized losses in use as compared to other cars.

The second rating corresponds to the degree to which the auto is more risky than its price indicates. Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

The third factor is the relative average loss payment per insured vehicle year. This value is normalized for all autos within a particular size classification (two-door small, station wagons, sports/specialty, etc.) and represents the average loss per car per year.

Given below is an embedded Excel Sheet of the **AUTOMOBILE DATASET**:



System Requirements and Software used:

The whole code has been written in ***PYTHON.***

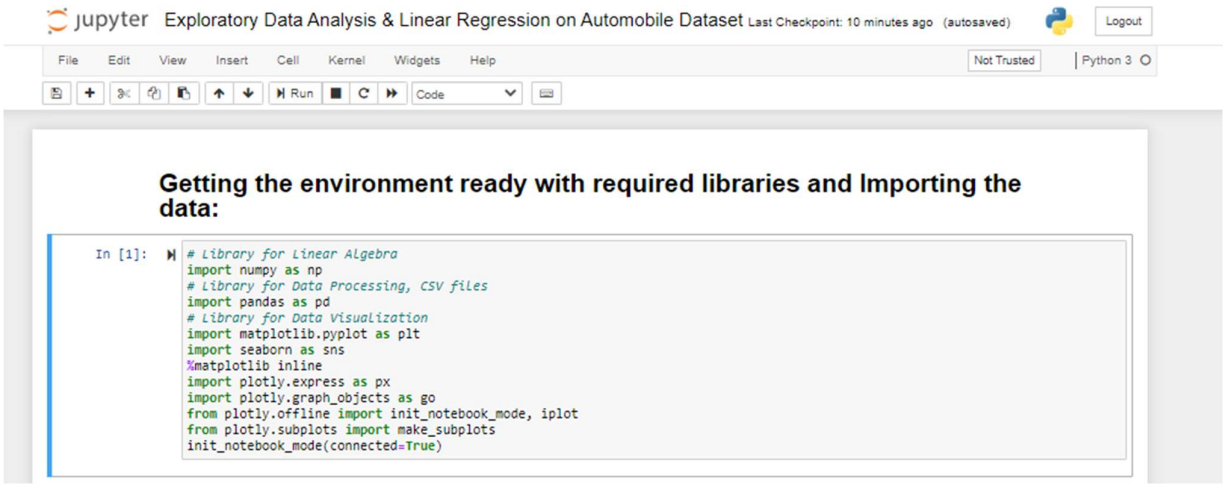
Software that has been used to develop and write the code is Jupyter Notebook. Anaconda software has been used to import and use all the libraries and packages for Machine Learning.

Libraries/Tools used:

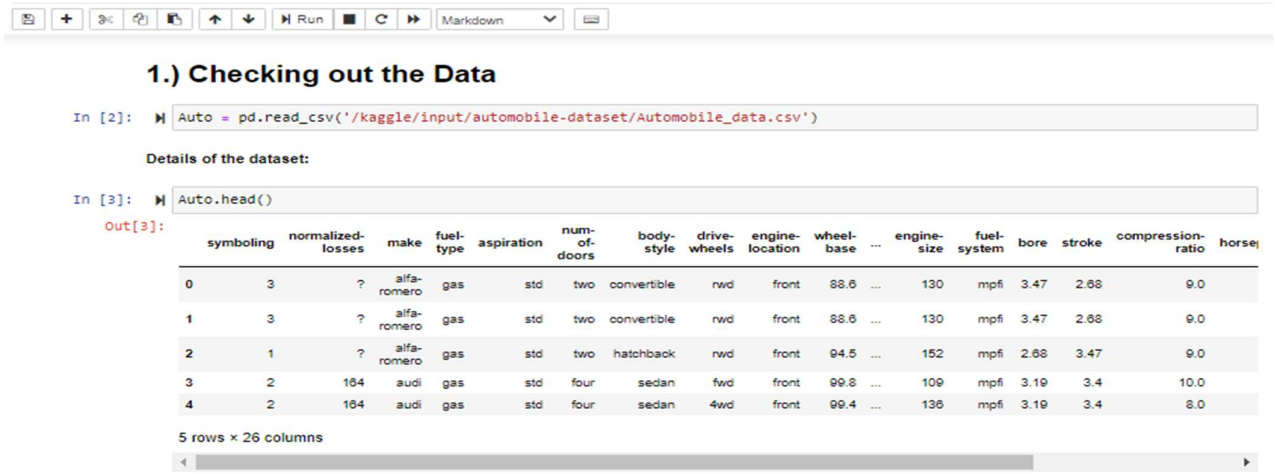
- Numpy
- Pandas
- Matplotlib
- Seaborn
- Plotly
- Scipy
- Scikit

Details of the project developed & Code:

Importing the data and required libraries:



1. Checking out the Data:



	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horse
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	8.0	

5 rows × 26 columns

```
In [4]: Auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    object
 19  stroke             205 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object
 22  peak-rpm           205 non-null    object
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              205 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

```
In [5]: Auto.describe()
```

Out[5]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

2. Handling the Missing Data:

Before analyzing and visualizing the data, we look for the missing values in the dataset. From the dataset it is noticed that there are few missing values, which are represented using special characters. The first aim is to replace all the special characters to NaN and then to choose the best method in removing the missing values.

2.) Handling missing data

In [24]:
```
# Let's convert the special characters to NaN
missing_values = ['?','--','-','??','.']

# new dataframe after replacing special characters from the set
Auto = Auto.replace(missing_values, np.nan)
Auto.head()
```

Out[24]:

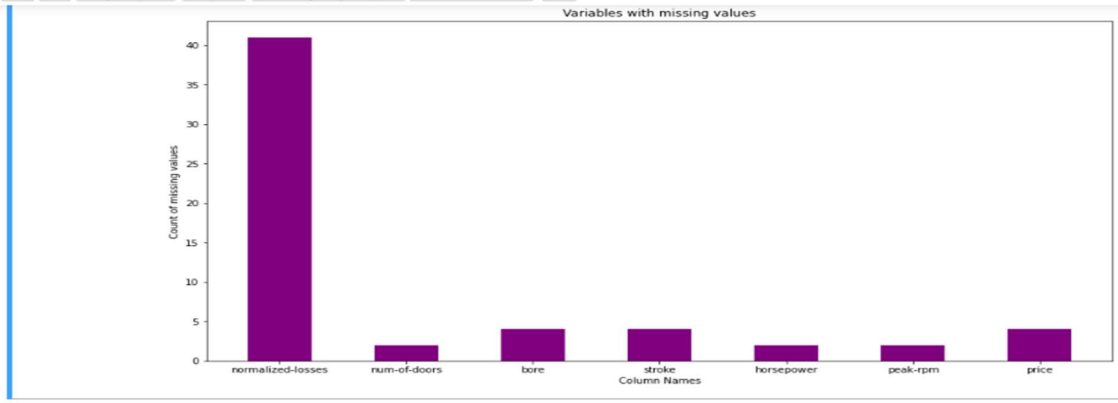
	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horse
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	8.0	

5 rows × 26 columns

In [7]:
```
# Let us look at the total missing values in the data set
# Looking for any missing values in the dataframe column
miss_val = Auto.columns[Auto.isnull().any()]

# printing out the columns and the total number of missing values of all the column
for column in miss_val:
    print(column, Auto[column].isnull().sum())
```

```
normalized-losses 41
num-of-doors 2
bore 4
stroke 4
horsepower 2
peak-rpm 2
price 4
```

In [8]:
```
# defining two empty lists for columns and its values
nan_columns = []
nan_values = []

for column in miss_val:
    nan_columns.append(column)
    nan_values.append(Auto[column].isnull().sum())

# plotting the graph
fig, ax = plt.subplots(figsize=(15,8))
plt.bar(nan_columns, nan_values, color = 'purple', width = 0.5)
ax.set_xlabel("Column Names")
ax.set_ylabel("Count of missing values")
ax.set_title("Variables with missing values");
```



From the above bar plot, there are totally 7 features with missing values. Taking a close look at the values, it is evident that all the missing feature variables can't be extracted from other available features but number of doors. We could get the value of **no of doors**, from features like **make** and **body-style**. For the temporal series of data, since we have outliers which lie an abnormal distance from other values in a random sample from population, it is recommendable to use median value to fill these missing data.

```
In [9]:
# Lets take the median to fill the missing values for the following features.
# In this process, we use the limit direction as both.

# Normalized losses
median_value= Auto['normalized-losses'].median()
Auto['normalized-losses']=Auto['normalized-losses'].fillna(median_value)

# Bore
median_value= Auto['bore'].median()
Auto['bore']=Auto['bore'].fillna(median_value)

# Stroke
median_value= Auto['stroke'].median()
Auto['stroke']=Auto['stroke'].fillna(median_value)

# Horsepower
median_value= Auto['horsepower'].median()
Auto['horsepower']=Auto['horsepower'].fillna(median_value)

# Peak-RPM
median_value= Auto['peak-rpm'].median()
Auto['peak-rpm']=Auto['peak-rpm'].fillna(median_value)

# Price
median_value= Auto['price'].median()
Auto['price']=Auto['price'].fillna(median_value)
```

Now let's move on to the next feature (num-of-doors), which is predictable from the available features (make and body-style). In this case let us look at the make and body style corresponding to the missing variable, num-of-doors. We already know there are 2 values missing in the column for num-of-doors feature.

```
In [10]:
# Looking for what the body_style and 'make' our missing values have
Auto[['make','body-style']][Auto['num-of-doors'].isnull()==True]
```

Out[10]:

	make	body-style
27	dodge	sedan
63	mazda	sedan

We have found that there are two missing values for the num-of-doors column. The body style and make is also clear. In order to decide the missing value, let us look at the num of doors corresponding to the sedan model of both dodge and Mazda.

```
In [11]:
# Looking for number of doors for a sedan model of Mazda
Auto['num-of-doors'][(Auto['body-style']=='sedan') & (Auto['make']=='mazda')]
```

```
Out[11]: 53    four
         54    four
         60    four
         62    four
         63     NaN
         65    four
         66    four
         Name: num-of-doors, dtype: object
```

```
In [12]:
# Similarly, looking for number of doors for a sedan model of Dodge
Auto['num-of-doors'][(Auto['body-style']=='sedan') & (Auto['make']=='dodge')]
```

```
Out[12]: 25    four
         26    four
         27     NaN
         Name: num-of-doors, dtype: object
```

From the two results, it is clear that for a sedan body style, there are 4 doors. So we now replace the Nan value to 4.

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

# dictionary mapping for num of doors
a=Auto['num-of-doors'].map({'two':2,'four':4})
Auto['num-of-doors']=a
```

```
In [14]:
# converting data type to int
Auto['num-of-doors'] = Auto['num-of-doors'].astype(str).astype(int)
```

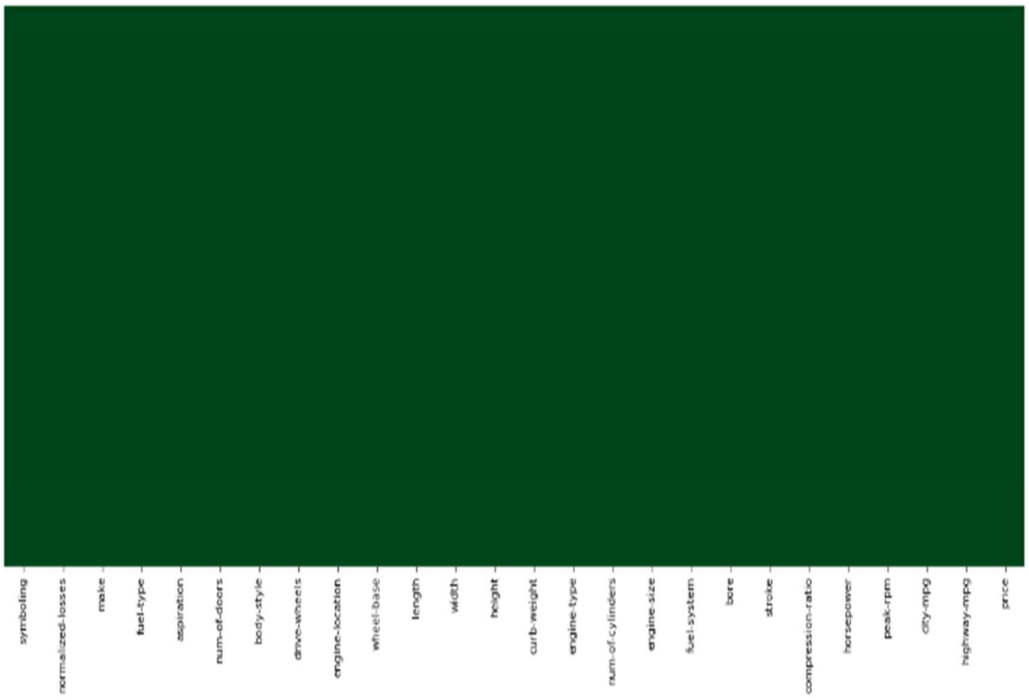
In [15]:
```
Auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    int64
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    object
 19  stroke             205 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object
 22  peak-rpm           205 non-null    object
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              205 non-null    object
dtypes: float64(5), int64(6), object(15)
memory usage: 41.8+ KB
```

From the above codes, we have filled up all those missing values, with meaningful values for analysis. But let's check it before proceeding to the next step.

In [16]:
```
# Heatmap

plt.subplots(figsize=(20,8))
sns.heatmap(Auto.isnull(),yticklabels=False,cbar=False,cmap='Greens_r')
```

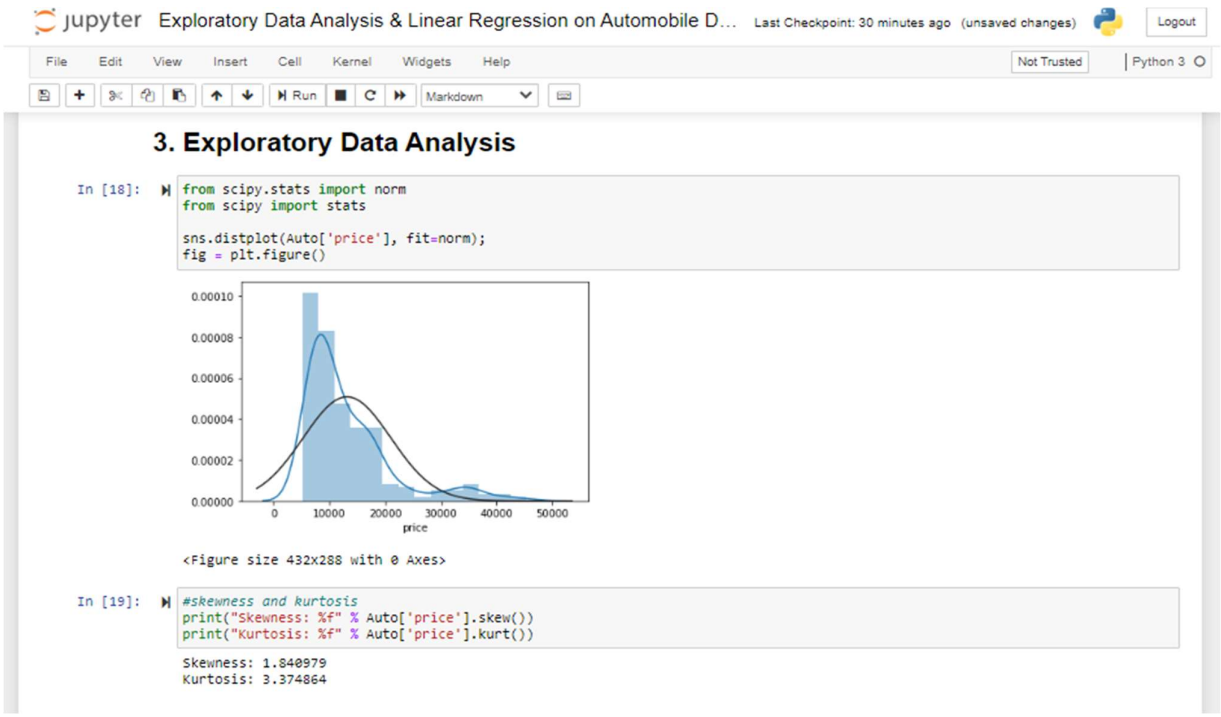
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe599a91210>



```
In [17]: Auto.isna().sum()

Out[17]: symboling           0
         normalized-losses   0
         make                0
         fuel-type           0
         aspiration          0
         num-of-doors        0
         body-style          0
         drive-wheels        0
         engine-location     0
         wheel-base          0
         length              0
         width               0
         height              0
         curb-weight         0
         engine-type         0
         num-of-cylinders    0
         engine-size         0
         fuel-system         0
         bore                0
         stroke              0
         compression-ratio   0
         horsepower          0
         peak-rpm            0
         city-mpg            0
         highway-mpg         0
         price               0
         dtype: int64
```

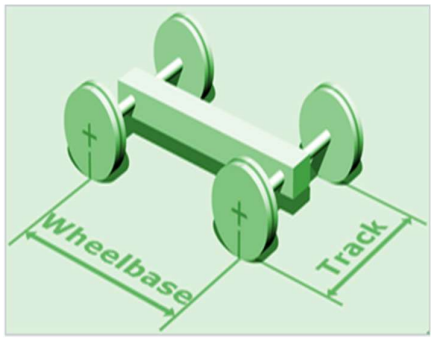
3. Exploratory Data Analysis (EDA):

EDA is a practice of iteratively asking a series of questions about the data at your hand and trying to build hypotheses based on the insights you gain from the data.

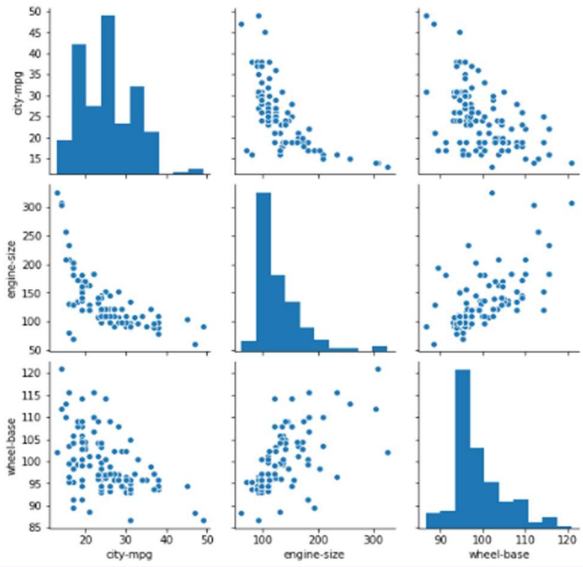


The given data is deviating from normal distribution, with positive skewness and with a positive kurtosis value, we have a leptokurtic distribution.

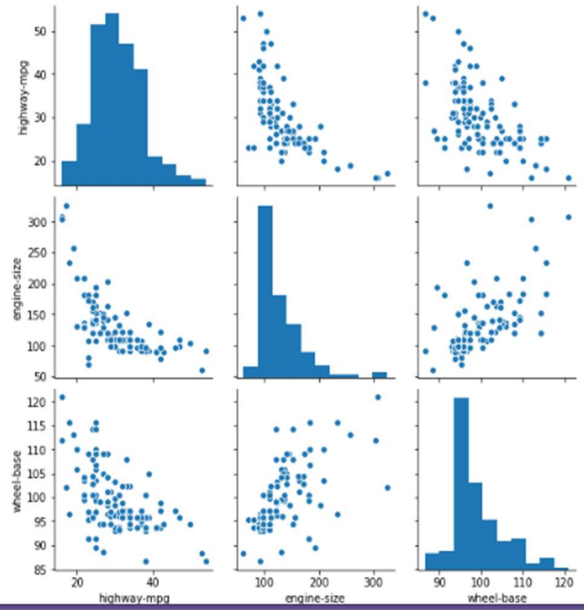
Let's look at the relationship between engine size, wheel base and mileage of the car.
Here is a brief idea about the features we plot here:
1. Engine size: The size of an engine is measured in cubic centimeters (cc) and refers to the total volume of air and fuel that's pushed through the engine by its cylinders. For example, a 1,000cc engine has the capacity to displace one liter - or 1,000 cubic centimeters - of this air-fuel mixture.
2. Wheelbase: It is the horizontal distance between the centers of the front and rear wheels.




```
In [20]: sns.pairplot(Auto[['city-mpg', 'engine-size', 'wheel-base']], palette='Set1')
Out[20]: <seaborn.axisgrid.PairGrid at 0x7fe59996c110>
```



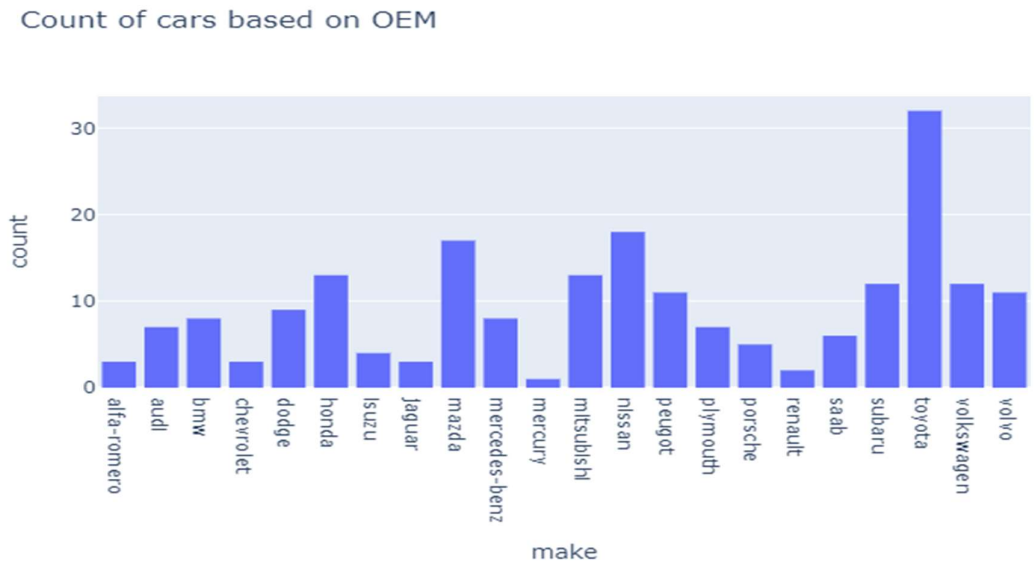
```
In [21]: sns.pairplot(Auto[['highway-mpg', 'engine-size', 'wheel-base']], palette='Set1')
Out[21]: <seaborn.axisgrid.PairGrid at 0x7fe5974a9490>
```



In both the cases, it is important to note that the mileage of car is inversely proportional to the engine size and wheel base. Even though a larger engine is more powerful, they generally use up more fuel than smaller ones. Similarly when the wheel base is big, then the mileage drops.

Now let us look at the car make, which OEM produces the more number of cars.

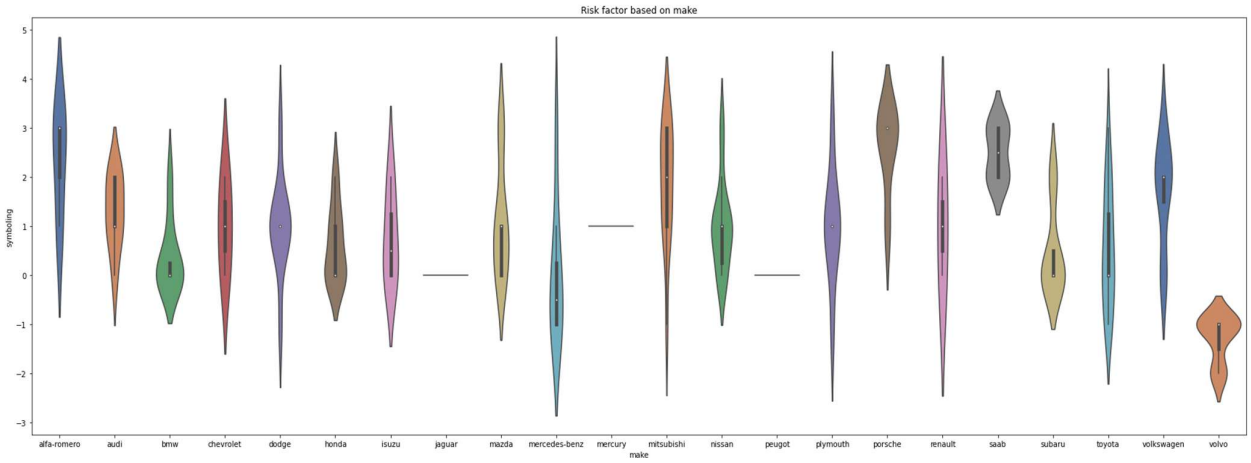
```
In [26]: fig = px.histogram(Auto, x="make", title='Count of cars based on OEM')
         fig.show()
```



Conclusion: Toyota is the leading producer of cars, followed by Nissan and then Mazda.

Risk-factor study: Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

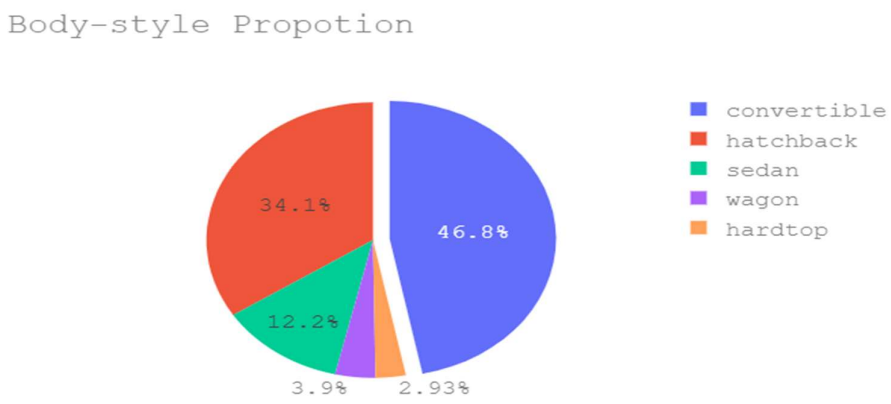
```
In [23]: # Comparing Symboling and make

         fig, ax = plt.subplots(figsize=(30,10))
         sns.violinplot(x="make", y="symboling", data=Auto, palette='deep')

         plt.title("Risk factor based on make")

Out[23]: Text(0.5, 1.0, 'Risk factor based on make')
```



From this plot, it is quite clear that Volvo outperform other OEMs, regarding the safety of the cars produced.

Now we could take a look at the body style for the cars:

```
In [24]: label = Auto["body-style"].unique()
         sizes = Auto["body-style"].value_counts().values

         # Now we could define the Pie chart
         # pull is given as a fraction of the pie radius. This serves the same purpose as explode
         fig_pie1 = go.Figure(data=[go.Pie(labels=label, values=sizes, pull=[0.1, 0, 0, 0])])
         # Defining the layout
         fig_pie1.update_layout(title="Body-style Propotion",
                 font=dict(
                 family="Courier New, monospace",
                 size=18,
                 color="#7f7f7f"
             ))
         fig_pie1.show()
```




```
In [25]: # Plotting multiple violinplots, including a box and scatter diagram
         fig_vio1 = px.violin(x = Auto['body-style'], y = Auto["price"], box=True, points="all")
         # Defining the layout
         fig_vio1.update_layout(
             title="Body-style and Price",
             xaxis_title="Body-style",
             yaxis_title="Price",
             font=dict(
                 family="Courier New, monospace",
                 size=18
             ))
         fig_vio1.show()
```

From the visualization we found out that convertible is the most produced type. However the price for hardtop is more than the convertible, making it the most expensive body type in the class

Knowing the best racer:

Looking for the fastest-accelerating car:

Adding a new column representing Torque:

Here in the dataset we have no column representing the torque. However we could calculate the Torque from the Horsepower and the RPM, using the following equation

Torque = (Horsepower x 5252) ÷ RPM.

In [26]:
```
# since the datatype of horsepower and peak-rpm is object, let's convert it first into integer
Auto['horsepower'] = Auto['horsepower'].astype(int)
Auto['peak-rpm'] = Auto['peak-rpm'].astype(int)

# let's create the new column for torque
Auto['torque'] = ((Auto['horsepower'] * 5252) / Auto['peak-rpm'])
Auto.head()
```

Out[26]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system	bore	stroke	compression-ratio	horsepower	pe r
0	3	115	alfa-romero	gas	std	2	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	5(
1	3	115	alfa-romero	gas	std	2	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	5(
2	1	115	alfa-romero	gas	std	2	hatchback	rwd	front	94.5	...	mpfi	2.68	3.47	9.0	154	5(
3	2	164	audi	gas	std	4	sedan	fwd	front	99.8	...	mpfi	3.19	3.4	10.0	102	5(
4	2	164	audi	gas	std	4	sedan	4wd	front	99.4	...	mpfi	3.19	3.4	8.0	115	5(

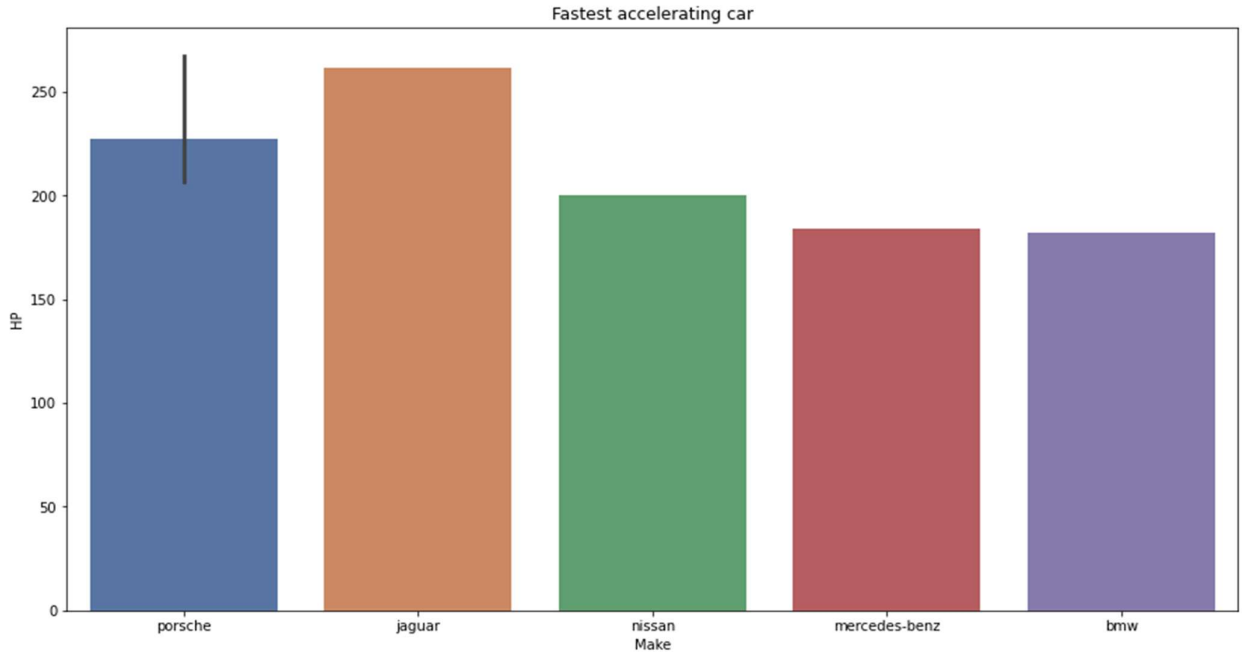
5 rows × 27 columns

In [27]:
```
# Extracting the first 10 largest Horsepower values
HP = Auto['horsepower'].nlargest(10)
HP_Index = [129,49,126,127,128,105,73,74,15,16]
# Extracting the corresponding 10 values from the column 'make'
make_hp = Auto['make'].iloc[HP_Index]

#creating a new dataframe with this values.
data = {'HP': HP, 'Make':make_hp}
df = pd.DataFrame(data)
df
```

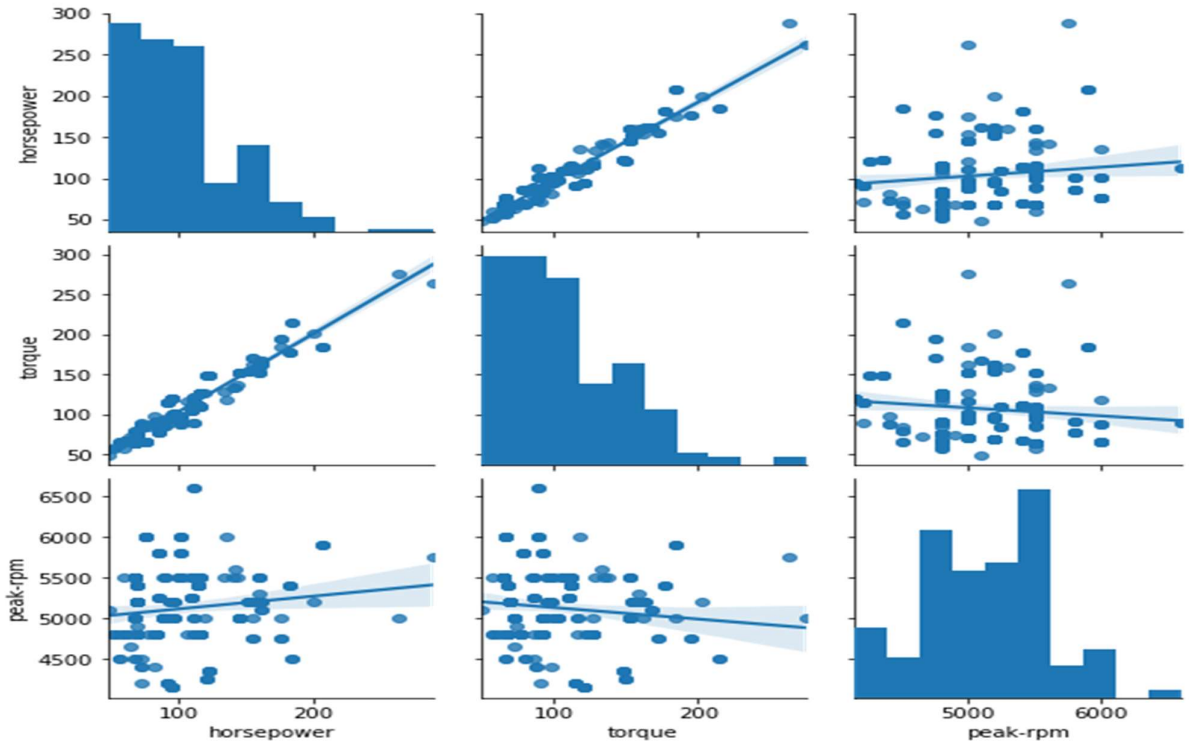
Out[27]:

	HP	Make
129	288	porsche
49	262	jaguar
126	207	porsche
127	207	porsche
128	207	porsche
105	200	nissan
73	184	mercedes-benz
74	184	mercedes-benz
15	182	bmw
16	182	bmw

```
In [28]: plt.subplots(figsize=(15,8))
         ax = sns.barplot(x="Make", y="HP", data=df, palette='deep')
         ax.set_xlabel("Make")
         ax.set_ylabel("HP")
         ax.set_title("Fastest accelerating car");
```




```
In [29]: sns.pairplot(Auto[['horsepower','torque','peak-rpm']], palette='Set1', kind="reg")

Out[29]: <seaborn.axisgrid.PairGrid at 0x7fe599c4de90>
```

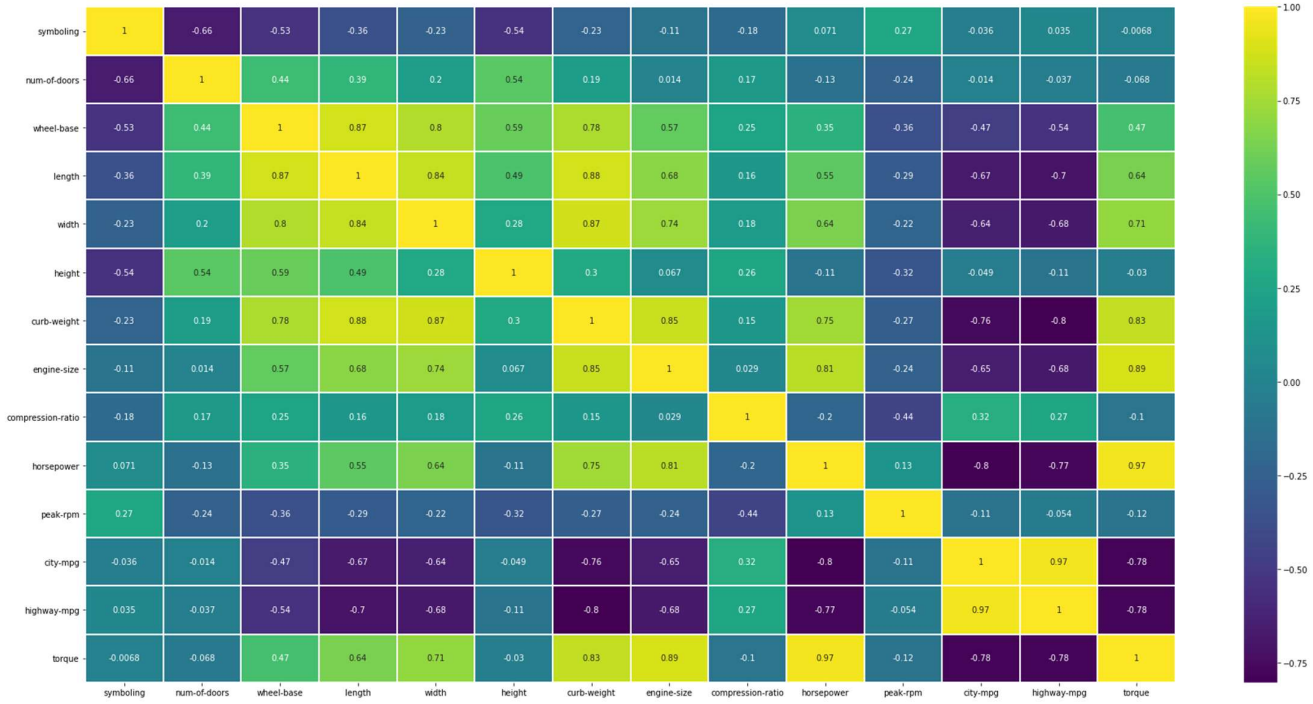


Conclusion:
Porsche has the highest Horsepower and so, it is the fastest accelerated car, followed by Jaguar.

4. Correlation between different features:

The correlation is calculated by using a heat map.

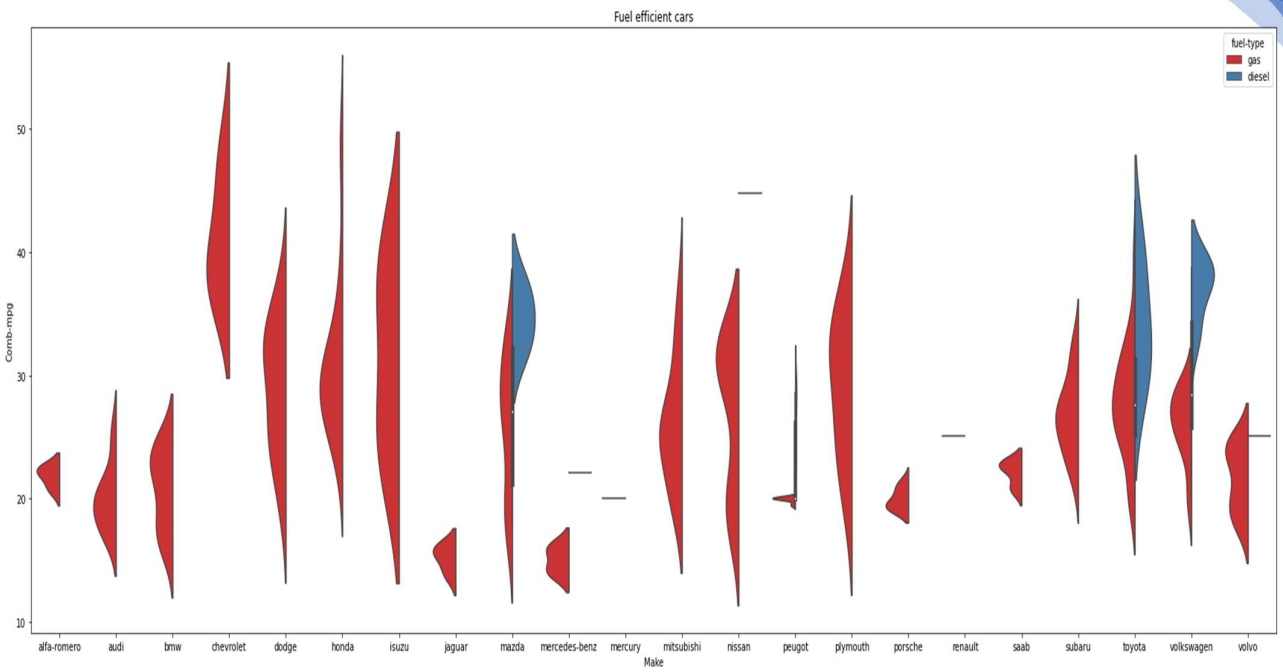
```
In [30]: plt.figure(figsize=(30,15))
         sns.heatmap(Auto.corr(),annot=True,cmap='viridis',linecolor='white',linewidths=1);
```



From the Correlation matrix it is evident that there are variables that are strongly correlated like city mpg and highway mpg. Likewise, there are variables that are negatively correlated too, like torque and mpg.
The Combined MPG value is the most prominent for the purpose of quick and easy comparison across vehicles. Combined fuel economy is a weighted average of City and Highway MPG values that is calculated by weighting the City value by 55% and the Highway value by 45%.
We could calculate the combined fuel economy from the available mpg values.

```
In [31]: # Calculating the combined mpg and creating a new dataframe
         Comb_mpg = ((Auto['highway-mpg'] * 0.4) + (Auto['city-mpg'] * 0.55))
         data1 = {'comb-mpg':Comb_mpg, 'make':Auto['make'], 'fuel-type':Auto['fuel-type']}

         df1 = pd.DataFrame(data1) # for the easiness of visualising
```

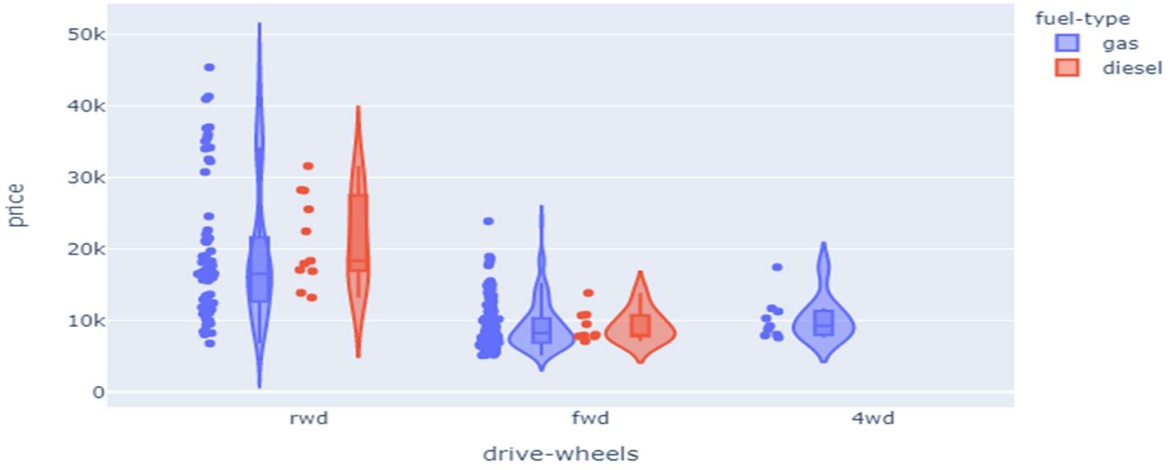
```
In [32]: # We use a violin plot mwith hue as the fuel type to know which car is fuel efficient

         #fuel_effi = df1.nlargest(5, ['comb-mpg'])  ---------- to know the first most fuel efficient cars
         #fuel_effi

         plt.subplots(figsize=(30,10))
         vx = sns.violinplot(x="make", y="comb-mpg", data=df1,hue='fuel-type',split=True,palette='Set1')
         vx.set_xlabel("Make")
         vx.set_ylabel("Comb-mpg")
         vx.set_title("Fuel efficient cars");
```



From the above plot, we could see that Japanese based Honda is the most fuel efficient car, followed by the American brand **Chevrolet**. The second and third runner ups are again other Japanese brands, **Nissan** and **Toyota** respectively. It is also noticeable that the Petrol engine cars are much efficient than its counterpart (Diesel) by a margin close to 4 mpg.

Now moving on to the **drive wheels** of the cars, let us look at the expensive category in this:

```
In [33]: fig = px.violin(Auto, y="price", x="drive-wheels", box=True, points="all",
                         color='fuel-type', hover_data=Auto.columns)
         fig.show()
```



Conclusion: The visualization says the rear wheel drive is costly in make, regardless whether it is petrol or diesel. This is because, getting power to the rear wheels need putting up the long driveshaft, fixing differential in the back and then connecting wheels to differentials just like in Front wheel drive. This would endure extra cost even during maintenance as well.

5. Data Cleaning:

Data cleansing or **data cleaning** is the process of detecting and correcting corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Data cleansing may be performed interactively with data wrangling tools, or as batch processing through scripting.

The first step would be to convert categorical features to dummy variables, else the machine learning algorithm won't be able to directly take in those features as inputs.

5. Data Cleaning:

In [34]:
```
Auto.head()
```

Out[34]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system	bore	stroke	compression-ratio	horsepower	p
0	3	115	alfa-romero	gas	std	2	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	5
1	3	115	alfa-romero	gas	std	2	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	5
2	1	115	alfa-romero	gas	std	2	hatchback	rwd	front	94.5	...	mpfi	2.68	3.47	9.0	154	5
3	2	164	audi	gas	std	4	sedan	fwd	front	99.8	...	mpfi	3.19	3.4	10.0	102	5
4	2	164	audi	gas	std	4	sedan	4wd	front	99.4	...	mpfi	3.19	3.4	8.0	115	5

5 rows × 27 columns

In [35]:
```
Auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 27 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    int64
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    object
 19  stroke             205 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    int64
 22  peak-rpm           205 non-null    int64
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              205 non-null    object
 26  torque             205 non-null    float64
dtypes: float64(6), int64(8), object(13)
memory usage: 43.4+ KB
```

In [36]:
```
#Let's split fuel type, drive wheels and engine location

fuel_type = pd.get_dummies(Auto['fuel-type'], drop_first=True)
# this would remove the diesel column after spliting the fuel-type column, which is automatically predictable

drive_wheels = pd.get_dummies(Auto['drive-wheels'], drop_first=True)
# this would remove the 4wd column after spliting the drive-wheels column, which is automatically predictable

engine_location = pd.get_dummies(Auto['engine-location'], drop_first=True)
# this would remove the front column after spliting the engine-location column, which is automatically predictable

aspiration = pd.get_dummies(Auto['aspiration'], drop_first=True)
# this would remove the std column after spliting the aspiration column, which is automatically predictable
```

In [37]:
```
Auto.drop(['fuel-type','drive-wheels','engine-location', 'aspiration'],axis=1,inplace=True)
```

In [38]:
```
Auto_df = pd.concat([Auto,fuel_type,drive_wheels,engine_location, aspiration], axis=1)
```

In [39]:
```
Auto_df.head()
```

Out[39]:

	symboling	normalized-losses	make	num-of-doors	body-style	wheel-base	length	width	height	curb-weight	...	peak-rpm	city-mpg	highway-mpg	price	torque	gas	fwd	rw
0	3	115	alfa-romero	2	convertible	88.6	168.8	64.1	48.8	2548	...	5000	21	27	13495	116.594400	1	0	
1	3	115	alfa-romero	2	convertible	88.6	168.8	64.1	48.8	2548	...	5000	21	27	16500	116.594400	1	0	
2	1	115	alfa-romero	2	hatchback	94.5	171.2	65.5	52.4	2823	...	5000	19	26	16500	161.761600	1	0	
3	2	164	audi	4	sedan	99.8	176.6	66.2	54.3	2337	...	5500	24	30	13950	97.400727	1	1	
4	2	164	audi	4	sedan	99.4	176.6	66.4	54.3	2824	...	5500	18	22	17450	109.814545	1	0	

5 rows × 28 columns

Few features which are considered important are sorted and the data type is converted to float

In [40]:
```
# converting the following data type to float
Auto_df['normalized-losses'] = Auto_df['normalized-losses'].astype(float)
Auto_df['bore'] = Auto_df['bore'].astype(float)
Auto_df['stroke'] = Auto_df['stroke'].astype(float)
Auto_df['price'] = Auto_df['price'].astype(float)
```

In [41]:
```
Auto_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 28 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    float64
 2   make               205 non-null    object
 3   num-of-doors       205 non-null    int64
 4   body-style         205 non-null    object
 5   wheel-base         205 non-null    float64
 6   length             205 non-null    float64
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   curb-weight        205 non-null    int64
 10  engine-type        205 non-null    object
 11  num-of-cylinders   205 non-null    object
 12  engine-size        205 non-null    int64
 13  fuel-system        205 non-null    object
 14  bore               205 non-null    float64
 15  stroke             205 non-null    float64
 16  compression-ratio  205 non-null    float64
 17  horsepower         205 non-null    int64
 18  peak-rpm           205 non-null    int64
 19  city-mpg           205 non-null    int64
 20  highway-mpg        205 non-null    int64
 21  price              205 non-null    float64
 22  torque             205 non-null    float64
 23  gas                205 non-null    uint8
 24  fwd                205 non-null    uint8
 25  rwd                205 non-null    uint8
 26  rear               205 non-null    uint8
 27  turbo              205 non-null    uint8
dtypes: float64(10), int64(8), object(5), uint8(5)
memory usage: 38.0+ KB
```

In order to create a linear regression model, we now to need to remove all irrelevant data. For this purpose, all the columns with 'object' datatype are removed.

In [42]:
```
Auto_df.select_dtypes(include='object')
```

Out[42]:

	make	body-style	engine-type	num-of-cylinders	fuel-system
0	alfa-romero	convertible	dohc	four	mpfi
1	alfa-romero	convertible	dohc	four	mpfi
2	alfa-romero	hatchback	ohcv	six	mpfi
3	audi	sedan	ohc	four	mpfi
4	audi	sedan	ohc	five	mpfi
...	...	...	...	...	...
200	volvo	sedan	ohc	four	mpfi
201	volvo	sedan	ohc	four	mpfi
202	volvo	sedan	ohcv	six	mpfi
203	volvo	sedan	ohc	six	idi
204	volvo	sedan	ohc	four	mpfi

205 rows × 5 columns

In [43]:
```
Auto_df.drop(['make','body-style','engine-type','num-of-cylinders', 'fuel-system'],axis=1,inplace=True)
```

In [44]:
```
Auto_clean = pd.concat([Auto_df],axis=1)
```

In [45]:
```
Auto_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 23 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    float64
 2   num-of-doors       205 non-null    int64
 3   wheel-base         205 non-null    float64
 4   length             205 non-null    float64
 5   width              205 non-null    float64
 6   height             205 non-null    float64
 7   curb-weight        205 non-null    int64
 8   engine-size        205 non-null    int64
 9   bore               205 non-null    float64
 10  stroke             205 non-null    float64
 11  compression-ratio  205 non-null    float64
 12  horsepower         205 non-null    int64
 13  peak-rpm           205 non-null    int64
 14  city-mpg           205 non-null    int64
 15  highway-mpg        205 non-null    int64
 16  price              205 non-null    float64
 17  torque             205 non-null    float64
 18  gas                205 non-null    uint8
 19  fwd                205 non-null    uint8
 20  rwd                205 non-null    uint8
 21  rear               205 non-null    uint8
 22  turbo              205 non-null    uint8
dtypes: float64(10), int64(8), uint8(5)
memory usage: 30.0 KB
```

6. Building Linear Regression Model:

The dataset available is split into training and test set. We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the price column**.**

6. Building Linear Regression Model:

```
In [46]: from sklearn.model_selection import train_test_split

In [47]: X = Auto_clean.drop('price', axis=1) # This would consider all the columns except price
         y = Auto_clean['price']

In [48]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

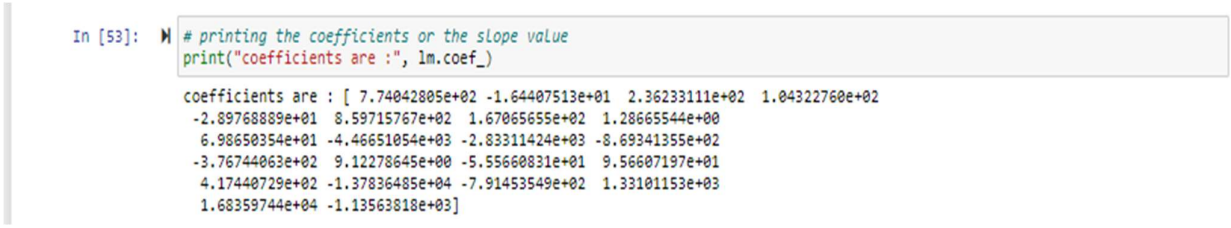
Training and Predicting data

```
In [49]: from sklearn.linear_model import LinearRegression

In [50]: lm = LinearRegression()

In [51]: lm.fit(X_train,y_train)
Out[51]: LinearRegression()
```

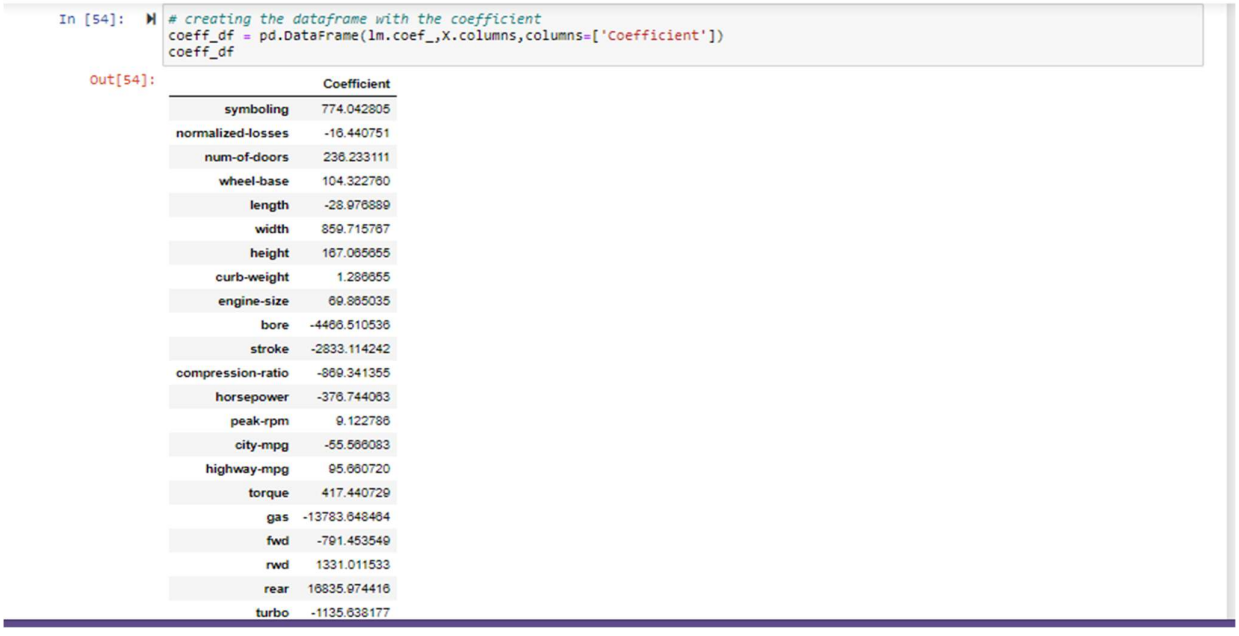
Model Evaluation

Let's evaluate the model by checking out it's coefficients and how we can interpret them.

```
In [52]: # printing the y intercept
         print("y intercept is :", lm.intercept_)
         y intercept is : -77848.30046447451
```



```
In [53]: # printing the coefficients or the slope value
         print("coefficients are :", lm.coef_)

coefficients are : [ 7.74042805e+02 -1.64407513e+01  2.36233111e+02  1.04322760e+02
 -2.89768889e+01  8.59715767e+02  1.67065655e+02  1.28665544e+00
  6.98650354e+01 -4.46651054e+03 -2.83311424e+03 -8.69341355e+02
 -3.76744063e+02  9.12278645e+00 -5.55660831e+01  9.56607197e+01
  4.17440729e+02 -1.37836485e+04 -7.91453549e+02  1.33101153e+03
  1.68359744e+04 -1.13563818e+03]
```



```
In [54]: # creating the dataframe with the coefficient
         coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
         coeff_df
```

Out[54]:

	Coefficient
symboling	774.042805
normalized-losses	-16.440751
num-of-doors	236.233111
wheel-base	104.322760
length	-28.976889
width	859.715767
height	167.065655
curb-weight	1.286655
engine-size	69.865035
bore	-4466.510536
stroke	-2833.114242
compression-ratio	-869.341355
horsepower	-376.744063
peak-rpm	9.122786
city-mpg	-55.566083
highway-mpg	95.660720
torque	417.440729
gas	-13783.648464
fwd	-791.453549
rwd	1331.011533
rear	16835.974416
turbo	-1135.638177

Interpreting the coefficients:

Let's look at how the following features affect the price of the car.

- Holding all other features fixed, a 1 unit increase in engine-size is associated with an *increase of $76.828156* in the price
- Holding all other features fixed, a 1 unit increase in city-mpg is associated with an *increase of $29.058026* in the price

7. Predictions from our Model:

7. Predictions from our Model:

In [55]:
```
predictions = lm.predict(X_test)
```

In [56]:
```
# Lets look at the predictions
predictions
```

Out[56]:
```
array([ 6692.47034307, 13062.14767098,  6244.19811593,  9496.19155902,
       11076.07017127, 15123.46456544,  9496.19155902, 14224.9602907 ,
       16365.68869182,  6974.85962517,  7116.68471635,  9510.54828987,
       11649.34943686, 19585.18058121,  7565.08715011,  8083.93397094,
       20197.30256444, 17711.41342211, 10148.47368715, 11295.96200767,
       15509.09215739,  8067.49321963, 19979.48935726, 27373.52175287,
       16306.80855823, 23732.49259164, 15668.78212648,  7680.54731086,
        9089.24006326, 21004.53252644, 15247.98465338,  9991.78018721,
        5818.65773168, 18016.12166158, 15084.16249912,  5942.3636329 ,
        7159.88567462,  7935.64287803, 11109.10657696,  6738.95408612,
       19770.13147149, 10127.88720007,  8640.03240728,  9442.33706897,
       10490.53715217, 22296.29667754, 19922.51433696, 19699.36542216,
        9776.96989137,  7855.77549718, 10255.81453678, 13589.08630516,
        6851.74725297,  7201.92990771,  7506.61767129,  8012.84220458,
       13660.93279575, 15791.80885316,  6703.91897989, 26931.07992895,
       22834.64723035, 11456.48751435])
```

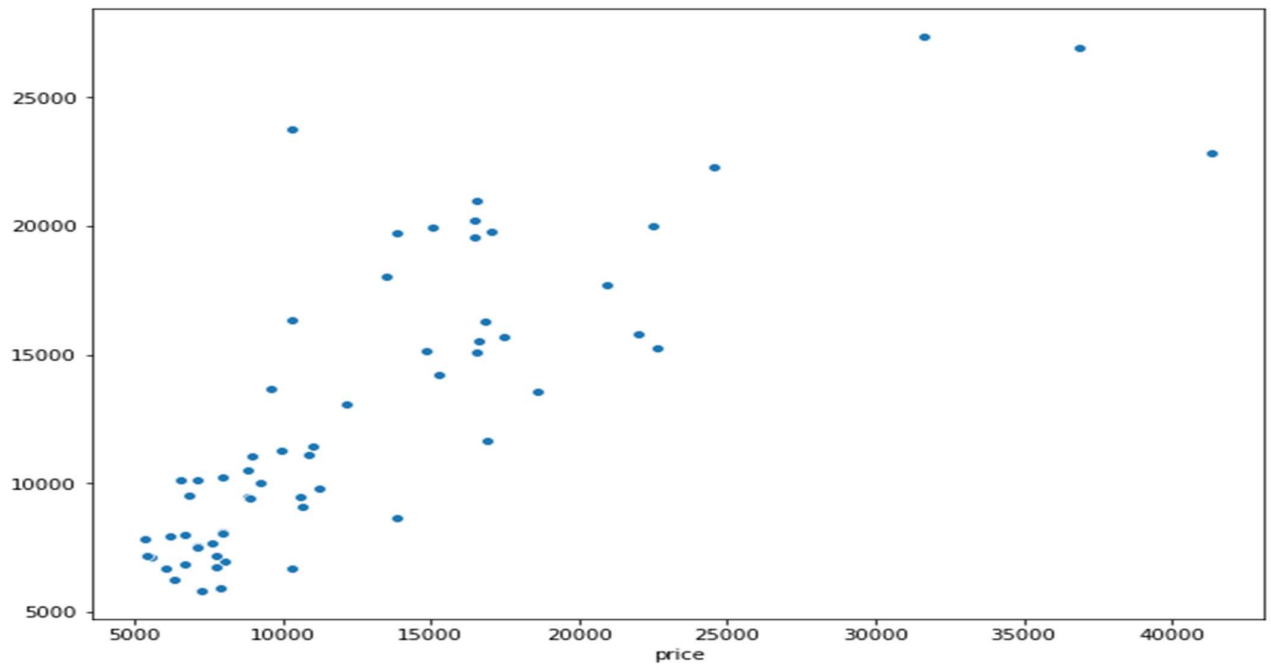
In [57]:
```
# now let us look at the y_test
y_test
```

Out[57]:
```
51      6095.0
133    12170.0
151     6338.0
61     10595.0
173     8948.0
        ...
125    22018.0
45     10295.0
17     36880.0
16     41315.0
46     11048.0
Name: price, Length: 62, dtype: float64
```

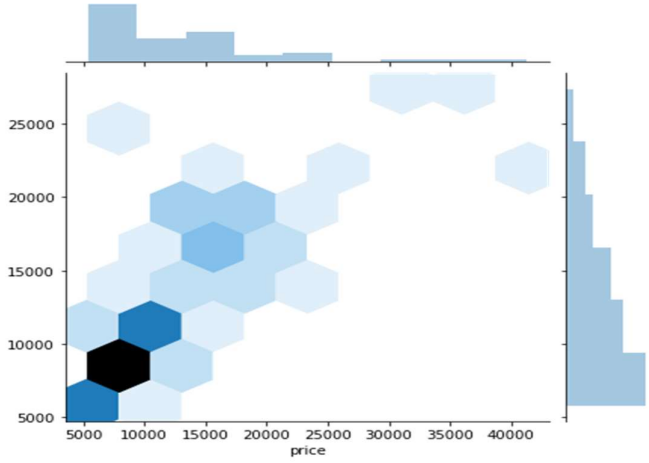
Residuals

Let's quickly explore the residuals to make sure everything was okay with our data.

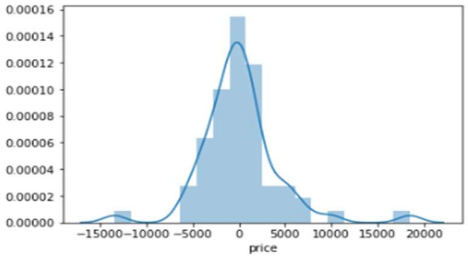
In [58]:
```
plt.subplots(figsize=(10,8))
ax = sns.scatterplot(x=y_test, y=predictions,
                     sizes=(20, 200), legend="full", palette="Set2")
```



```
In [59]: # Residuals are the difference between the actual and the predicted values
         sns.jointplot(x=y_test,y=predictions,kind='hex')

Out[59]: <seaborn.axisgrid.JointGrid at 0x7fe58ca08c10>
```




```
In [60]: # ploting the prediction error
         pred_error = y_test - predictions

         sns.distplot((pred_error))

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe596061c90>
```



Using Statsmodel OLS for Regression

```
In [61]: # Linear regression using Statsmodel
         import statsmodels.api as sm

         # Unlike sklearn that adds an intercept to our data for the best fit, statsmodel doesn't. We need to add it ourselves
         # Remember, we want to predict the price based off our features.
         # X represents our predictor variables, and y our predicted variable.
         # We need now to add manually the intercepts

         X_endog = sm.add_constant(X_train)
```

```
In [62]: # fitting the model
         ls = sm.OLS(y_train, X_endog).fit() # OLS = Ordinary Least Squares
         # summary of the model
         ls.summary()
```

t[62]:

OLS Regression Results

Dep. Variable:	price	R-squared:	0.914
Model:	OLS	Adj. R-squared:	0.898
Method:	Least Squares	F-statistic:	58.00
Date:	Fri, 17 Jul 2020	Prob (F-statistic):	1.87e-53
Time:	14:30:35	Log-Likelihood:	-1314.1
No. Observations:	143	AIC:	2674.
Df Residuals:	120	BIC:	2742.
Df Model:	22		
Covariance Type:	nonrobust		

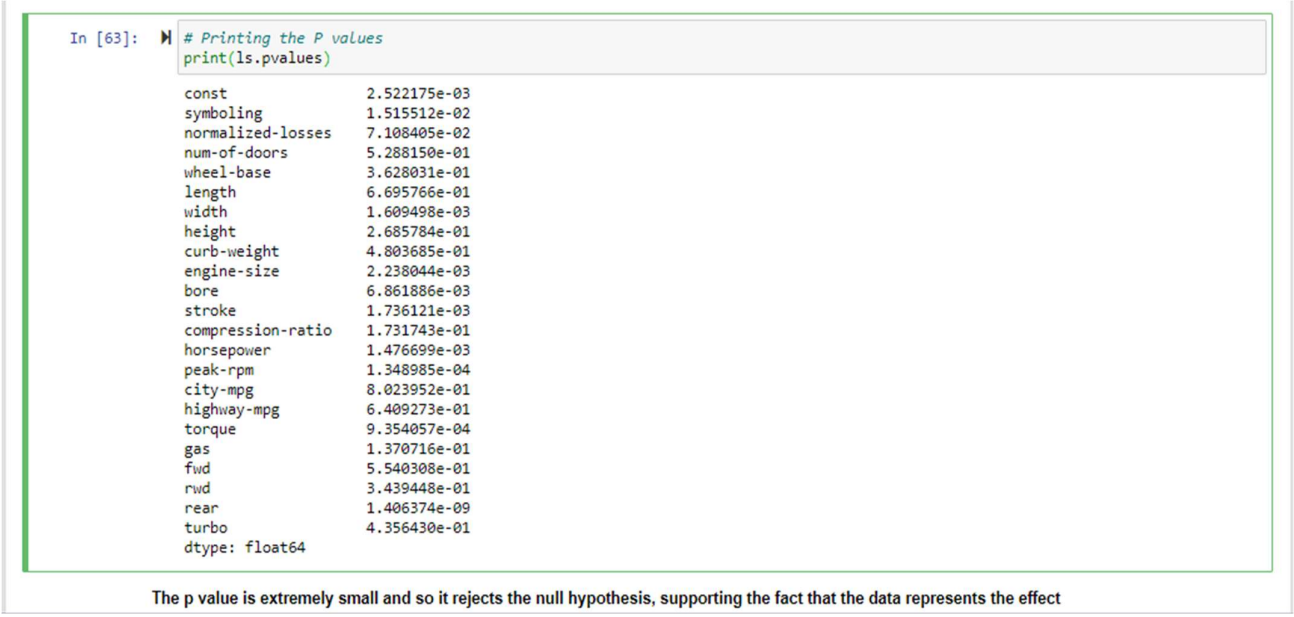
	coef	std err	t	P>|t|	[0.025	0.975]
const	-7.785e+04	2.52e+04	-3.086	0.003	-1.28e+05	-2.79e+04
symboling	774.0428	314.136	2.464	0.015	152.075	1396.011
normalized-losses	-16.4408	9.028	-1.821	0.071	-34.315	1.434
num-of-doors	236.2331	373.992	0.632	0.529	-504.245	976.711
wheel-base	104.3228	114.199	0.914	0.363	-121.784	330.429
length	-28.9769	67.738	-0.428	0.670	-163.092	105.139
width	859.7158	266.360	3.228	0.002	332.342	1387.089
height	167.0657	150.308	1.111	0.269	-130.533	464.664
curb-weight	1.2867	1.818	0.708	0.480	-2.312	4.885
engine-size	69.8650	22.365	3.124	0.002	25.584	114.146
bore	-4466.5105	1623.573	-2.751	0.007	-7681.073	-1251.948
stroke	-2833.1142	884.237	-3.204	0.002	-4583.842	-1082.386
compression-ratio	-869.3414	634.451	-1.370	0.173	-2125.510	386.827
horsepower	-376.7441	115.766	-3.254	0.001	-605.952	-147.536
peak-rpm	9.1228	2.313	3.945	0.000	4.544	13.702
city-mpg	-55.5661	221.553	-0.251	0.802	-494.225	383.093
highway-mpg	95.6607	204.583	0.468	0.641	-309.400	500.722
torque	417.4407	123.010	3.394	0.001	173.890	660.992
gas	-1.378e+04	9208.759	-1.497	0.137	-3.2e+04	4449.054
fwd	-791.4535	1333.762	-0.593	0.554	-3432.210	1849.303
rwd	1331.0115	1400.841	0.950	0.344	-1442.556	4104.579
rear	1.684e+04	2565.109	6.563	0.000	1.18e+04	2.19e+04
turbo	-1135.6382	1451.868	-0.782	0.436	-4010.236	1738.959

Omnibus:	3.213	Durbin-Watson:	2.263
Prob(Omnibus):	0.201	Jarque-Bera (JB):	2.694
Skew:	0.279	Prob(JB):	0.260
Kurtosis:	3.374	Cond. No.	6.86e+05

8. Calculating the P-Value:

The P Value basically helps to answer the question: 'Does the data really represent the observed effect?'. The P Value is the probability of seeing the effect (E) when the null hypothesis is true. A sufficiently low value is required to reject the null hypothesis. So, when the p-value is low enough, we reject the null hypothesis and conclude the observed effect holds. This level of 'low enough' cutoff is called the alpha level, and you need to decide it before conducting a statistical test.

Alpha level is the cutoff probability for p-value to establish statistical significance for a given hypothesis test. For an observed effect to be considered as statistically significant, the p-value of the test should be lower than the pre-decided alpha value. Typically for most statistical tests (but not always), alpha is set as 0.05. But when the occurrence of the event is rare, you may want to set a very low alpha. The rarer it is, the lower the alpha. Whereas for a more likely event, it can go up to 0.1.



```
In [63]: # Printing the P values
         print(ls.pvalues)

const                2.522175e-03
symboling            1.515512e-02
normalized-losses    7.108405e-02
num-of-doors         5.288150e-01
wheel-base           3.628031e-01
length               6.695766e-01
width                1.609498e-03
height               2.685784e-01
curb-weight          4.803685e-01
engine-size          2.238044e-03
bore                 6.861886e-03
stroke               1.736121e-03
compression-ratio    1.731743e-01
horsepower           1.476699e-03
peak-rpm             1.348985e-04
city-mpg             8.023952e-01
highway-mpg          6.409273e-01
torque               9.354057e-04
gas                  1.370716e-01
fwd                  5.540308e-01
rwd                  3.439448e-01
rear                 1.406374e-09
turbo                4.356430e-01
dtype: float64
```

The p value is extremely small and so it rejects the null hypothesis, supporting the fact that the data represents the effect

9. Regression Evaluation Metrics:

Here are the common evaluation metrics for regression problems:

- **Mean Absolute Error (MAE)** is the mean of the absolute value of the errors
- **Mean Squared Error (MSE)** is the mean of the squared errors
- **Root Mean Squared Error (RMSE)** is the square root of the mean of the squared errors:
- **R-squared (R2)** indicates the percentage of the variance in the dependent variable that the independent variables explain collectively

Comparing these metrics:

1. MAE is the easiest to understand, because it's the average error.
2. MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
3. RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.
4. R-Squared could be interpreted so -
    - 0 : doesn't fit the data
    - 1 : better fit for the data
    - -1 : due to overfitting of data

All of these are loss functions, because we want to minimize them.

In [64]:
```
from sklearn import metrics
```

In [65]:
```
# Mean Absolute Error (MAE)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))

# Mean Squared Error (MSE)
print('MSE:', metrics.mean_squared_error(y_test, predictions))

# Root Mean Squared Error (RMSE)
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

# R-squared (R2)
print('R^2 Score:', metrics.r2_score(y_test, predictions))
```

```
MAE: 2798.6143215693514
MSE: 17969702.58088184
RMSE: 4239.068598275079
R^2 Score: 0.6637911262316374
```

-----The End-----