“A PROJECT REPORT ON PRICE PREDICTION OF USED CARS”

SUBMITTED BY

HIMAJA IJJADA

ACKNOWLEDGMENT

I express my sincere gratitude to FlipRobo Technologies for giving me the opportunity to work on the price prediction of used cars project using machine learning algorithms. I would also like to thank FlipRobo Technologies for providing me with the requisite knowledge to webscrape the datasets to work with. And I would like to express my gratitude to Mr. Mohd Kashif (SME FlipRobo) and Ms. Sapna Verma (SME FlipRobo) for being of a great help in completion of the project.

Most of the concepts used to predict the Prices of used cars project are learned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/

CONTENTS

Introduction

▪ Business Problem Framing

Predicting the price of used cars is an important and interesting problem. Predicting the resale value of a car is not a simple task. It is trite knowledge that the value of used cars depends on a number of factors. The most important ones are usually the age of the car, its make (model), the origin of the car (the original location of the manufacturer), its mileage (the number of kilometers it has run) and its horsepower (amount of power that an engine produces). Due to rising fuel prices, fuel economy is also of prime importance. Unfortunately, in practice, most people do not know exactly how much fuel their car consumes for each km driven. Other factors such as the type of fuel it uses, the interior style, the braking system, acceleration, engine displacement, the volume of its cylinders (measured in cc), its size, number of doors, paint color, weight of the car, consumer reviews, prestigious awards won by the car manufacturer, its physical state, whether it is a sports car, whether it has cruise control, whether it is automatic or manual transmission, whether it belonged to an individual or a company and other options such as air conditioner, sound system, power steering, cosmic wheels, GPS navigator all may influence the price as well. Some special factors which buyers attach importance is the local of previous owners, whether the car had been involved in serious accidents. The look and feel of the car certainly contribute a lot to the price. As we can see, the price depends on a large number of factors. Unfortunately, information about all these factors are not always available and the buyer must make the decision to purchase at a certain price based on few factors only. In this work, we have considered only a small subset of the factors which are more important.

With the Covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. One of our clients works with small traders, who sell used cars. With the change in market due to Covid 19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data. We have to make car price valuation model.

Business goal: The main aim of this project is to predict the price of used car based on various features. Machine Learning is a field of technology developing with immense abilities and applications in automating tasks. So, we will deploy an ML model for car selling price prediction and analysis. This kind of system becomes handy for many people. This model will provide the approximate selling price for the car based on different features like fuel type, transmission, and price, weight, running in kms, engine displacement, mileage etc. and this model will help the client to understand the price of used cars.

▪ Conceptual Background of the Domain Problem

Car Price Prediction is really an interesting machine learning problem as there are many factors that influence the price of a car in the second-hand market. In many developed countries, it is common to lease a car rather than buying it outright. A lease is a binding contract between a buyer and a seller (or a third party – usually a bank, insurance firm or other financial institutions) in which the buyer must pay fixed instalments for a pre-defined number of months/years to the seller/financer. After the lease period is over, the buyer has the possibility to buy the car at its residual value, i.e., its expected resale value. Thus, it is of commercial interest to seller/financers to be able to predict the salvage value (residual value) of cars with accuracy. If the residual value is under-estimated by the seller/financer at the beginning, the instalments will be higher for the clients who will certainly then opt for another seller/financer. If the residual value is over-estimated, the instalments will be lower for the clients but then the seller/financer may have much difficulty at selling these high-priced used cars at this over-estimated residual value. Thus, we can see that estimating the price of used cars is of very high commercial importance as well.

Here we are trying to help the client works with small traders, who sell used cars to understand the price of the used cars by deploying machine learning models. These models would help the client/sellers to understand the used car market and accordingly they would be able to sell the used car in the market.

▪ Review of Literature

Literature review covers relevant literature with the aim of gaining insight into the factors that are important to predict the price of used cars in the market. In this study, we discuss various applications and methods which inspired us to build our supervised ML techniques to predict the price of used cars in different locations. We did a background survey regarding the basic ideas of our project and used those ideas for the collection of data information by doing web scraping from www.cardekho.com website which is a web platform where seller can sell their used car.

This project is more about data exploration, feature engineering and pre-processing that can be done on this data. Since we scrape huge amount of data that includes more car related features, we can do better data exploration and derive some interesting features using the available columns. Different techniques like ensemble techniques, k-nearest neighbors, and decision trees have been used to make the predictions.

The goal of this project is to build an application which can predict the car prices with the help of other features. In the long term, this would allow people to better explain and reviewing their purchase with each other in this increasing digital world.

▪ Motivation for the Problem Undertaken

Deciding whether a used car is worth the posted price when you see listings online can be difficult. Several factors, including mileage, engine displacement, running, make, model, year, etc. can influence the actual worth of a car. From the perspective of a seller, it is also a dilemma to price a used car appropriately.

So, the main aim is to use machine learning algorithms to develop models for predicting used car prices.

- To build a supervised machine learning model for forecasting value of a vehicle based on multiple attributes.
- The system that is being built must be feature based i.e., feature wise prediction must be possible.
- Providing graphical comparisons to provide a better view

Analytical problem framing

▪ Mathematical/ Analytical Modeling of the Problem

We need to develop an efficient and effective Machine Learning model which predicts the price of a used cars. So, “Car_Price” is our target variable which is continuous in nature. Clearly it is a Regression problem where we need to use regression algorithms to predict the results.

This project is done on two phases:

I. **Data Collection Phase:** I have done web scraping to collect the data of used cars from the well-known website www.cardekho.com where I found more features of cars compared to other websites and I fetch data for different locations. As per the requirement of our client we need to build the model to predict the prices of these used cars.

II. **Model Building Phase:** After collecting the data, I built a machine learning model. Before model building, have done all data pre-processing steps. The complete life cycle of data science that I have used in this project are as follows:
- Data Cleaning
- Exploratory Data Analysis
- Data Pre-processing
- Model Building
- Model Evaluation
- Selecting the best model

▪ Data Sources and their formats

We have collected the dataset from the website www.cardekho.com which is a web platform where seller can sell their used car. The data is scrapped using Web scraping technique and the framework used is Selenium. We scrapped nearly 12600 of the data and fetched the data for different locations and collected the information of different features of the car and saved the collected data in excel format. The dimension of the dataset is 12608 rows and 20 columns including target variable “Car_Price”.

```
#importing dataset
df = pd.read_excel("Used_Cars.xlsx") #Reading excel file
#df.head()
```

▪ Data Preprocessing Done

Data pre-processing is the process of converting raw data into a well-readable format to be used by Machine Learning model. Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model. I have used following pre-processing steps:

Dropping unnecessary column

```
#Droping unnecessary column
df = df.drop(["Unnamed: 0"],axis=1)
```

Checking shape of the dataset

```
#Checking shape of the dataset
df.shape
```

```
(12608, 20)
```

The dataset has 12608 rows and 20 columns.

Checking all column names

```
#Checking all column names
df.columns
```

```
Index(['Car_Name', 'Fuel_type', 'Running_in_kms', 'Engine_disp',
       'Gear_transmission', 'Milage_in_km/ltr', 'Seating_cap', 'color',
       'Max_power', 'front_brake_type', 'rear_brake_type', 'cargo_volume',
       'height', 'width', 'length', 'Weight', 'Insp_score', 'top_speed',
       'City_url', 'Car_price'],
      dtype='object')
```

Above are the list of column names in the dataset.

Feature description:

- Car_Name : Name of the car with Year
- Fuel_type : Type of fuel used for car engine
- Running_in_kms : Car running in kms till the date
- Engine_disp : Engine displacement/engine CC
- Gear_transmission : Type of gear transmission used in car
- Milage_in_km/ltr : Overall milage of car in Km/ltr
- Seating_cap : Availability of number of seats in the car
- color : Car color
- Max_power : Maximum power of engine used in car in bhp

- front_brake_type : type of brake system used for front-side wheels
- rear_brake_type : type of brake system used for back-side wheels
- cargo_volume : the total cubic feet of space in a car's cargo area.
- height : Total height of car in mm
- width : Width of car in mm
- length : TOtal length of the car in mm
- Weight : Gross weight of the car in kg
- Insp_score : inspection rating out of 10
- top_speed : Maximum speed limit of the car in km per hours
- City_url : Url of the page of cars from a particular city
- Car_price : Price of the car

Checking for missing values

```
#Checking for missing values
df.isnull().sum()
```

```
Car_Name              0
Fuel_type             0
Running_in_kms        0
Engine_disp           0
Gear_transmission     0
Milage_in_km/ltr      0
Seating_cap          55
color                 0
Max_power             1
front_brake_type     76
rear_brake_type      76
cargo_volume        447
height               56
width                56
length               56
Weight               37
Insp_score            0
top_speed          1798
City_url              0
Car_price             0
dtype: int64
```

There are some entries like '-' and 'null' so let's replace these with nan.

Replacing unnecessary entries with nan

```
#Replacing unnecessary entries with nan
df.replace('-',np.nan, inplace = True)
df.replace('null ',np.nan, inplace = True)
```

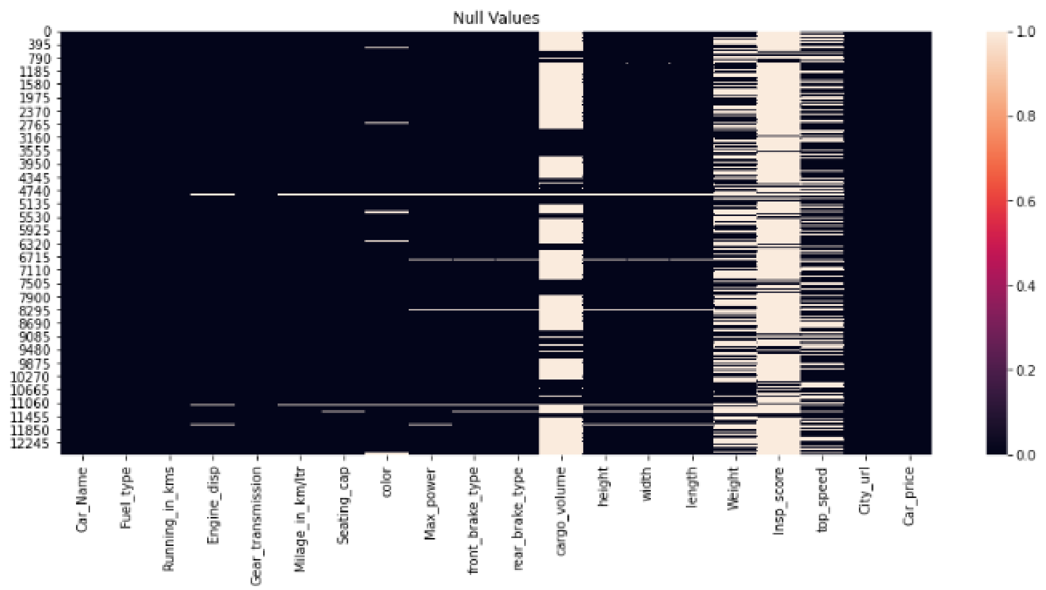
Checking for nan values again

```
#Checking for nan values again
df.isnull().sum()
```

```
Car_Name                 0
Fuel_type                0
Running_in_kms           0
Engine_disp             60
Gear_transmission        0
Milage_in_km/ltr        29
Seating_cap            104
color                  274
Max_power              145
front_brake_type       214
rear_brake_type        215
cargo_volume          8388
height                 254
width                  255
length                 254
Weight                6074
Insp_score           10876
top_speed             4316
City_url                 0
Car_price                0
dtype: int64
```

Visualizing null values

```
#Visualizing null values
plt.figure(figsize=[15,6])
sns.heatmap(df.isnull())
plt.title("Null Values")
plt.show()
```



Observations

By visualization of null values we can clearly say that

- There are huge null values in the following columns of the dataset-
  - cargo_volume
  - Weight
  - Insp_score
  - top_speed
- There are few missing values in the following columns
  - Engine_disp

- Milage_in_km/ltr
    - Seating_cap
    - color
    - Max_power
    - front_brake_type
    - rear_brake_type
    - height
    - width
    - length
- We can drop those columns with more than 50% missing values and others with less than 50% missing values can be replaced using iteration methods

Dropping columns with more than 50% of missiing values

```
#Dropping columns with more than 50% of missing values
df.drop(columns = ['cargo_volume','Insp_score','Weight','top_speed'], inplace = True)
```

We have dropped the above columns

Checking the info about the dataset

```
#Checking the info about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12608 entries, 0 to 12607
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Car_Name           12608 non-null  object
 1   Fuel_type          12608 non-null  object
 2   Running_in_kms     12608 non-null  object
 3   Engine_disp        12548 non-null  object
 4   Gear_transmission  12608 non-null  object
 5   Milage_in_km/ltr   12579 non-null  object
 6   Seating_cap        12504 non-null  object
 7   color              12334 non-null  object
 8   Max_power          12463 non-null  object
 9   front_brake_type   12394 non-null  object
 10  rear_brake_type    12393 non-null  object
 11  height             12354 non-null  object
 12  width              12353 non-null  object
 13  length             12354 non-null  object
 14  City_url           12608 non-null  object
 15  Car_price          12608 non-null  object
dtypes: object(16)
memory usage: 1.5+ MB
```

Observations

Above is the info about the dataset and we shall fill these missing values in the dataset using imputation methods.

Car_Name:

As the Car_name column has year of manufacture, car model and car name all together so we shall extract them for better seperated values

```
#Extracting manufacturing year and car name from Car_Name
df['Manu_year'] = df['Car_Name'].str[0:4]
df['car_name'] = df['Car_Name'].str[4:]
df.drop(columns = 'Car_Name', inplace = True)
```

```
df['Car_Brand'] = df.car_name.str.split(' ').str.get(1)
df['Car_Model'] = df.car_name.str.split(' ').str[2:]
df['Car_Model'] = df['Car_Model'].apply(lambda x: ','.join(map(str, x)))
df['Car_Model'] = df['Car_Model'].str.replace(',',' ')
df.drop(columns = 'car_name', inplace = True)
```

Car_Price:

Since Car_Price is our target, it shall be in the continuous data format. So we have to change the car_price column from lakhs and crores to integer format.

```
df['car_price'] = df['Car_price'].str.replace('Lakh','100000')
df['car_price'] = df['car_price'].str.replace(',','')
df['car_price'] = df['car_price'].str.replace('Cr','10000000')
```

```
df[['a','b']] = df.car_price.str.split(expand=True)
```

```
df['a'] = df['a'].astype('float')
df['b'] = df['b'].astype('float')
```

Now we have separated the column with alpha numeric data to replace the values with float type data

Checking for null values in b column

```
#Checking for null values in b column
df.isnull().sum()
```

```
Fuel_type             0
Running_in_kms        0
Engine_disp          60
Gear_transmission     0
Milage_in_km/ltr     29
Seating_cap         104
color               274
Max_power           145
front_brake_type    214
rear_brake_type     215
height              254
width               255
length              254
City_url              0
Car_price             0
Manu_year             0
Car_Brand             0
Car_Model             0
car_price             0
a                     0
b                    91
dtype: int64
```

```
df['b']=df['b'].fillna(value = 1)
```

We have filled the nan values in the column

```
df['car_price'] = df['a'] * df['b']
```

Now we have converted the data in car_price to float type data

```
df.drop(columns = ['Car_price','a','b'], inplace = True)
```

We have dropped the columns as they are redundant

```
df['Running_in_kms'] = df['Running_in_kms'].str.replace('kms','')
df['Running_in_kms'] = df['Running_in_kms'].str.replace(',','')
df['Running_in_kms'] = df['Running_in_kms'].str.replace('1 Lakh ','100000')
df['Running_in_kms'] = df['Running_in_kms'].astype('float')
```

Running_in_kms:

Since this column should be int datatype but it has some string values and ',' in between so let's replace them.

```
df['Running_in_kms'] = df['Running_in_kms'].str.replace('kms','')
df['Running_in_kms'] = df['Running_in_kms'].str.replace(',','')
df['Running_in_kms'] = df['Running_in_kms'].str.replace('1 Lakh ','100000')
df['Running_in_kms'] = df['Running_in_kms'].astype('float')
```

We have replaced the values in the above column with number of km and converted into the float type data

```
df.dtypes
```

```
Fuel_type              object
Running_in_kms        float64
Engine_disp            object
Gear_transmission      object
Milage_in_km/ltr       object
Seating_cap            object
color                  object
Max_power              object
front_brake_type       object
rear_brake_type        object
height                 object
width                  object
length                 object
City_url               object
Manu_year              object
Car_Brand              object
Car_Model              object
car_price             float64
dtype: object
```

Observations

- Most of the columns are in object type data.

Engine_disp:

The column 'Engine_disp' should be continuous column so we will convert it to float datatype.

```
df.Engine_disp = df.Engine_disp.astype('float')
```

we have converted the data type of 'Engine_disp' to float datatype.

Milage_in_km/ltr:

In Milage_in_km/ltr column the data type is object so we have to change this to float type.

```
df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].str.replace('kmpl','')
df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].str.replace('km/kg','')
df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].str.replace('km/hr','')

df['Milage_in_km/ltr'] = df['Milage_in_km/ltr'].astype('float')
```

We have converted the data type of 'Milage_in_km/ltr' to float datatype.

Converting the data type of columns height, width and length to float datatype:

```
df['height'] = df['height'].str.replace(',','')
df['height'] = df['height'].str[0:4]
df['width'] = df['width'].str.replace(',','')
df['length'] = df['length'].str.replace(',','')
df.height = df.height.astype('float')
df.width = df.width.astype('float')
df.length = df.length.astype('float')
```

we have converted the data type of columns height, width and length to float datatype

City_url:

Let's extract city name from city url column.

Checking value counts of City_url column

```
#Checking value counts of City_url column
df.City_url.value_counts()
```

```
https://www.cardekho.com/used-cars+in+delhi-ncr     1490
https://www.cardekho.com/used-cars+in+bangalore     1486
https://www.cardekho.com/used-cars+in+mumbai        1478
https://www.cardekho.com/used-cars+in+new-delhi     1473
https://www.cardekho.com/used-cars+in+pune          1239
https://www.cardekho.com/used-cars+in+gurgaon       1040
https://www.cardekho.com/used-cars+in+noida          982
https://www.cardekho.com/used-cars+in+hyderabad      918
https://www.cardekho.com/used-cars+in+chennai        836
https://www.cardekho.com/used-cars+in+kolkata        595
https://www.cardekho.com/used-cars+in+ahmedabad      579
https://www.cardekho.com/used-cars+in+jaipur         492
Name: City_url, dtype: int64
```

The above data shows the no of cars from each of the cities

Replacing city names from city urls

```
#Replacing city names from city urls
df['city_name'] = df.City_url.replace('https://www.cardekho.com/used-cars+in+bangalore', 'Bangalore')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+mumbai', 'mumbai')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+chennai', 'Chennai')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+hyderabad', 'hyderabad')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+pune', 'pune')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+delhi-ncr', 'delhi-ncr')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+ahmedabad', 'ahmedabad')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+gurgaon', 'gurgaon')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+noida', 'noida')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+kolkata', 'kolkata')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+jaipur', 'jaipur')
df['city_name'] = df.city_name.replace('https://www.cardekho.com/used-cars+in+new-delhi', 'new-delhi')
```

We have replaced the city names with city urls

Let's check the value count again

```
#Let's check the value count again
df['city_name'].value_counts()
```

```
delhi-ncr    1490
Bangalore    1486
mumbai       1478
new-delhi    1473
pune         1239
gurgaon      1040
noida         982
hyderabad     918
Chennai       836
kolkata       595
ahmedabad     579
jaipur        492
Name: city_name, dtype: int64
```

Since we have extracted city names let's drop City_url.

```
#Dropping unnecessary column
df.drop(columns = 'City_url', inplace = True)
```

We have dropped the city url column as it is redundant

Seating_cap:

Let's change the data type of seating_cap to float type.

```
#converting Seating_cap to float data type
df.Seating_cap = df.Seating_cap.astype('float')
```

Manu_Year:

Let's extract car age from manufactured year.

```
df.Manu_year = df.Manu_year.astype('float')
df['Car_age'] = 2021 - df['Manu_year']
df.drop(columns = 'Manu_year', inplace = True)
```

Max_power:

We have to change the datatype of Max_power column to float datatype.

Getting numerical values from column Max_power and converting them to float type

```
#getting numerical values from column Max_power and converting them to float type
df['Max_power'] = df['Max_power'].str[0:5]
```

```
df['Max_power'] = df['Max_power'].str.replace('PS','')
df['Max_power'] = df['Max_power'].str.replace('ps','')
df['Max_power'] = df['Max_power'].str.replace('Bh','')
df['Max_power'] = df['Max_power'].str.replace('P','')
```

```
df.Max_power = df.Max_power.astype('float')
```

We have changed the datatype of Max_power column to float datatype.

front_brake_type:

Let's group the similar entries in this column.

```
#Checking the value counts of front_brake_type
df['front_brake_type'].value_counts()
```

```
Disc                                     6902
Ventilated Disc                          4785
Solid Disc                                181
Ventilated Discs                          141
Disc & Caliper Type                        83
Disk                                       73
Ventilated DIsc                            51
Ventilated discs                           33
Drum                                       25
Ventilated Disk                            17
Multilateral Disc                          14
264mm Ventilated discs                     13
Electric Parking Brake                     11
Vantilated Disc                            10
Disc & Drum                                 7
Vacuum assisted hydraulic dual circuit w    7
Disc,internally ventilated                  6
Discs                                       6
Disc, 236 mm                                5
disc                                        4
Ventillated Disc                            4
Ventlated Disc                              4
Ventillated Discs                           3
Carbon ceramic                              2
Booster assisted ventilated disc            2
Tandem master cylinder with Servo assist    1
Dual Circuit with ABS, ABS with BAS         1
Ventilated disc                             1
Ventilated & Grooved Steel Discs            1
Mechanical-hydraulic dual circuit           1
Name: front_brake_type, dtype: int64
```

The 'front_brake_type' feature has few similar entries which shall be grouped for easy evaluation and better understanding

```
df["front_brake_type"].replace("Solid Disc","Disc",inplace=True)
df["front_brake_type"].replace("Disk","Disc",inplace=True)
df["front_brake_type"].replace("Discs","Disc",inplace=True)
df["front_brake_type"].replace("Disc, 236 mm","Disc",inplace=True)
df["front_brake_type"].replace("disc","Disc",inplace=True)

df["front_brake_type"].replace("Ventilated Discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated DIsc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated Disk","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("264mm Ventilated discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Vantilated Disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Disc,internally ventilated","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventlated Disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventillated Disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventillated Discs","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Booster assisted ventilated disc","Ventilated Disc",inplace=True)
df["front_brake_type"].replace("Ventilated disc","Ventilated Disc",inplace=True)
```

We have grouped all the similiar entries into one entry

Checking the value counts of front_brake_type again

```
#Checking the value counts of front_brake_type again
df['front_brake_type'].value_counts()
```

```
Disc                                          7171
Ventilated Disc                               5070
Disc & Caliper Type                             83
Drum                                            25
Multilateral Disc                               14
Electric Parking Brake                          11
Disc & Drum                                      7
Vacuum assisted hydraulic dual circuit w         7
Carbon ceramic                                   2
Ventilated & Grooved Steel Discs                 1
Dual Circuit with ABS, ABS with BAS              1
Tandem master cylinder with Servo assist         1
Mechanical-hydraulic dual circuit                1
Name: front_brake_type, dtype: int64
```

We can see that the value counts has reduced due to grouping of similar entries

rare_brake_type:

Let's group the similar entries in this column.

Checking value counts of rare_break_type column

```
#Checking value counts of rare_break_type column
df['rear_brake_type'].value_counts()
```

```
Drum                                          10022
Disc                                           1409
Ventilated Disc                                 296
Solid Disc                                      208
Leading-Trailing Drum                           103
Disc & Caliper Type                              83
Self-Adjusting Drum                              50
Discs                                            42
Ventilated discs                                 32
Ventilated Discs                                 25
Drums                                            20
262mm Disc & Drum Combination                    13
Disc & Drum                                      12
Self Adjusting Drum                              12
Electric Parking Brake                           11
Leading & Trailing Drum                           8
Ventilated Drum                                   8
Vacuum assisted hydraulic dual circuit w          7
Drums 180 mm                                      5
drum                                              4
Self Adjusting Drums                              3
Self adjusting Drums                              3
Drum in disc                                      2
Drum in Discs                                     2
Carbon ceramic                                    2
Booster assisted drum                             2
Ventialted Disc                                   2
Ventialte Disc                                    2
Self adjusting drums                              1
Dual Circuit with ABS, ABS with BAS               1
Ventilated & Grooved Steel Discs                  1
228.6 mm dia, drums on rear wheels                1
Mechanical-hydraulic dual circuit                 1
Name: rear_brake_type, dtype: int64
```

The 'rear_brake_type' feature has few similar entries which shall be grouped for easy evaluation and better understanding

```
df["rear_brake_type"].replace("Drums","Drum",inplace=True)
df["rear_brake_type"].replace("drum","Drum",inplace=True)
df["rear_brake_type"].replace("Drums 180 mm","Drum",inplace=True)
df["rear_brake_type"].replace("Drum in Discs","Drum",inplace=True)
df["rear_brake_type"].replace("Drum in disc","Drum",inplace=True)


df["rear_brake_type"].replace("Discs","Disc",inplace=True)
df["rear_brake_type"].replace("Solid Disc","Disc",inplace=True)
df["rear_brake_type"].replace("Disc Brakes","Disc",inplace=True)

df["rear_brake_type"].replace("Ventilated discs","Ventilated Disc",inplace=True)
df["rear_brake_type"].replace("Ventilated Discs","Ventilated Disc",inplace=True)
df["rear_brake_type"].replace("Ventialted Disc","Ventilated Disc",inplace=True)
df["rear_brake_type"].replace("Ventialte Disc","Ventilated Disc",inplace=True)

df["rear_brake_type"].replace("Leading & Trailing Drum","Leading-Trailing Drum",inplace=True)

df["rear_brake_type"].replace("Self Adjusting Drum","Self-Adjusting Drum",inplace=True)
df["rear_brake_type"].replace("Self Adjusting Drums","Self-Adjusting Drum",inplace=True)
df["rear_brake_type"].replace("Self adjusting Drums","Self-Adjusting Drum",inplace=True)
df["rear_brake_type"].replace("Self adjusting drums","Self-Adjusting Drum",inplace=True)
```

We have grouped all the similiar entries into one entry

checking the value count of each column

```
#Lets check the value count of each column to see if there are any unexpected and unwanted entries pres
for i in df.columns:
        print(df[i].value_counts())
        print('****************************************')
```

```
Petrol      7056
Diesel      5422
CNG           92
LPG           27
Electric      11
Name: Fuel_type, dtype: int64
****************************************
60000.0      142
65000.0      139
70000.0      138
80000.0      119
40000.0      106
50000.0      105
55000.0       98
45000.0       95
75000.0       94
35000.0       92
58000.0       85
68000.0       80
```

We can see that the value counts has reduced due to grouping of similar entries

Checking the datatypes of all columns after cleaning

```
#Checking the datatypes of all columns after cleaning
df.dtypes
```

```
Fuel_type              object
Running_in_kms        float64
Engine_disp           float64
Gear_transmission      object
Milage_in_km/ltr      float64
Seating_cap           float64
color                  object
Max_power             float64
front_brake_type       object
rear_brake_type        object
height                float64
width                 float64
length                float64
Car_Brand              object
Car_Model              object
car_price             float64
city_name              object
Car_age               float64
dtype: object
```

Now in the dataset we have two types of data - Float and object type data.

Checking unique values of each column

```
#Checking unique values of each column
df.nunique()
```

```
Fuel_type              5
Running_in_kms      4577
Engine_disp          143
Gear_transmission      2
Milage_in_km/ltr     478
Seating_cap            8
color                195
Max_power            376
front_brake_type      13
rear_brake_type       17
height               241
width                224
length               321
Car_Brand             35
Car_Model            275
car_price           1227
city_name             12
Car_age               25
dtype: int64
```

Above are the unique value counts of each column. We don't find anything unnecessary so let's proceed.

Imputation technique to replace nan values

```
#Checking null values in the dataset
df.isnull().sum()
```

```
Fuel_type             0
Running_in_kms        0
Engine_disp          60
Gear_transmission     0
Milage_in_km/ltr     29
Seating_cap         104
color               274
Max_power           145
front_brake_type    214
rear_brake_type     215
height              254
width               255
length              254
Car_Brand             0
Car_Model             0
car_price             0
city_name             0
Car_age               0
dtype: int64
```

We have to replace the nan values in continuous columns by there mean and categorical columns with it's mode.

```
#Checking for skewness in the dataset
df.skew()
```

```
Running_in_kms      7.906142
Engine_disp         1.895318
Milage_in_km/ltr   -0.511076
Seating_cap         2.444332
Max_power           2.939518
height              0.955643
width               0.800815
length              0.434823
car_price           9.610453
Car_age             0.723524
dtype: float64
```

In the numerical columns with skewness null values has to be replaced by median.

```
#Replacing nan values
for col in ['Engine_disp','Milage_in_km/ltr','Max_power','height','width']:
    df[col] = df[col].fillna(df[col].median())
for col in ['length']:
    df[col] = df[col].fillna(df[col].mean())
for col1 in ['Seating_cap','color','front_brake_type','rear_brake_type']:
    df[col1] = df[col1].fillna(df[col1].mode()[0])
```

We have replaced all the necessary values

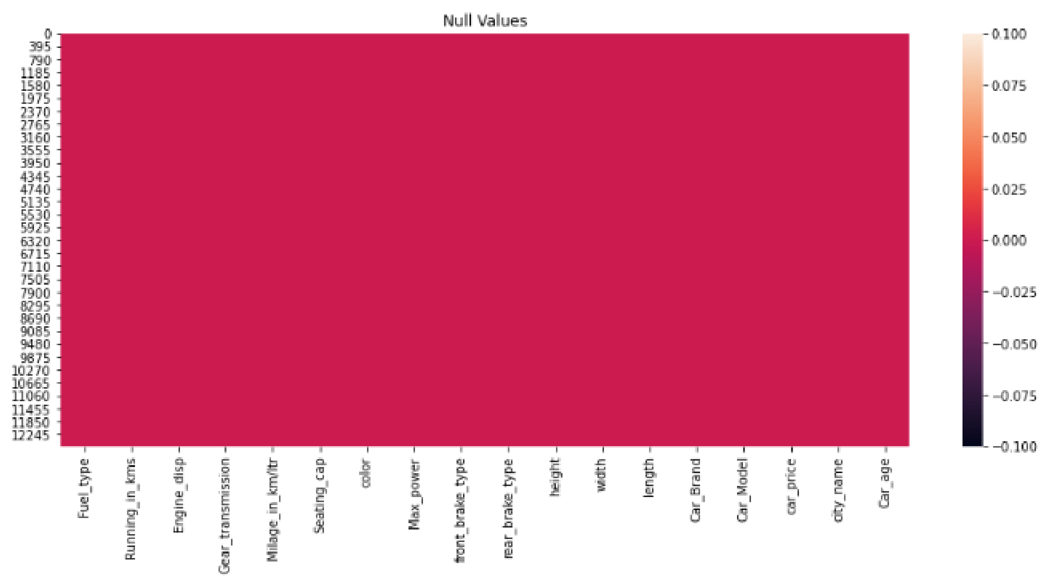
Checking null values in the dataset again

```
#Checking null values in the dataset again
df.isnull().sum()
```

```
Fuel_type           0
Running_in_kms      0
Engine_disp         0
Gear_transmission   0
Milage_in_km/ltr    0
Seating_cap         0
color               0
Max_power           0
front_brake_type    0
rear_brake_type     0
height              0
width               0
length              0
Car_Brand           0
Car_Model           0
car_price           0
city_name           0
Car_age             0
dtype: int64
```

Visualizing null values

```
#Visualizing null values
plt.figure(figsize=[15,6])
sns.heatmap(df.isnull())
plt.title("Null Values")
plt.show()
```



Observations

As we have replaced all null values successfully, there are no null values present in the dataset

▪ Data Inputs- Logic- Output Relationships

Printing the dataset

```
#Printing the dataset
df.head()
```

	Fuel_type	Running_in_kms	Engine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_t
0	Petrol	131125.0	998.0	Manual	21.79	5.0	Grey	67.05	
1	Petrol	73875.0	1197.0	Manual	18.90	5.0	White	82.00	
2	Diesel	97922.0	1498.0	Manual	22.27	5.0	White	108.60	Ventilated
3	Petrol	24230.0	998.0	Manual	21.70	5.0	Red	67.05	Ventilated
4	Petrol	41174.0	998.0	Automatic	20.51	5.0	Grey	67.00	Ventilated

```
#Checking description of data set
df.describe()
```

	Running_in_kms	Engine_disp	Milage_in_km/ltr	Seating_cap	Max_power	height	width	length
count	1.260800e+04	12608.000000	12608.000000	12608.000000	12608.000000	12608.000000	12608.000000	12608.000000
mean	5.772259e+04	1436.207249	19.556908	5.218036	100.130872	1563.792989	1718.849540	4083.963089
std	4.027723e+04	494.852497	4.220344	0.693750	44.445694	111.054497	125.361262	398.610518
min	2.000000e+02	0.000000	0.000000	2.000000	32.500000	148.000000	1410.000000	3099.000000
25%	3.300000e+04	1197.000000	17.010000	5.000000	74.000000	1488.000000	1675.250000	3765.000000
50%	5.500000e+04	1248.000000	19.600000	5.000000	86.800000	1520.000000	1700.000000	3995.000000
75%	7.586225e+04	1498.000000	22.070000	5.000000	113.400000	1630.000000	1765.000000	4413.000000
max	1.080000e+06	5998.000000	36.000000	10.000000	641.000000	1995.000000	2220.000000	5295.000000

▪ Assumptions

- All the columns have mean higher than the standard deviation. Standard deviation is a measure of how dispersed the data is in relation to the mean. Low standard deviation means data are clustered around the mean.
- Only the 'Milage_in_km/ltr' column have the 50% percentile higher than the mean, which says that the distribution is negatively skewed.
- Al the other columns have mean higher than the 50th percentile. When the mean is greater than the median, the distribution is positively skewed.
- There is a significant difference between the max value and 75th percentile for all the columns, which indicates presence of outliers.

▪ Hardware and Software Requirements and Tools Used

To build the machine learning projects it is important to have the following hardware and software requirements and tools.

Hardware required:

- Processor: core i5 or above
- RAM: 8 GB or above
- ROM/SSD: 250 GB or above

Software required:

- Distribution: Anaconda Navigator
- Programming language: Python
- Browser based language shell: Jupyter Notebook
- Chrome: To scrape the data

Model/s Development and evaluation

- ## Visualizations

Univariate Analysis:

```
# checking for categorical columns
categorical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_columns.append(i)
print(categorical_columns)
```

```
['Fuel_type', 'Gear_transmission', 'color', 'front_brake_type', 'rear_brake_type', 'Car_Brand', 'Car_
Model', 'city_name']
```

Above is the list of categorical columns.

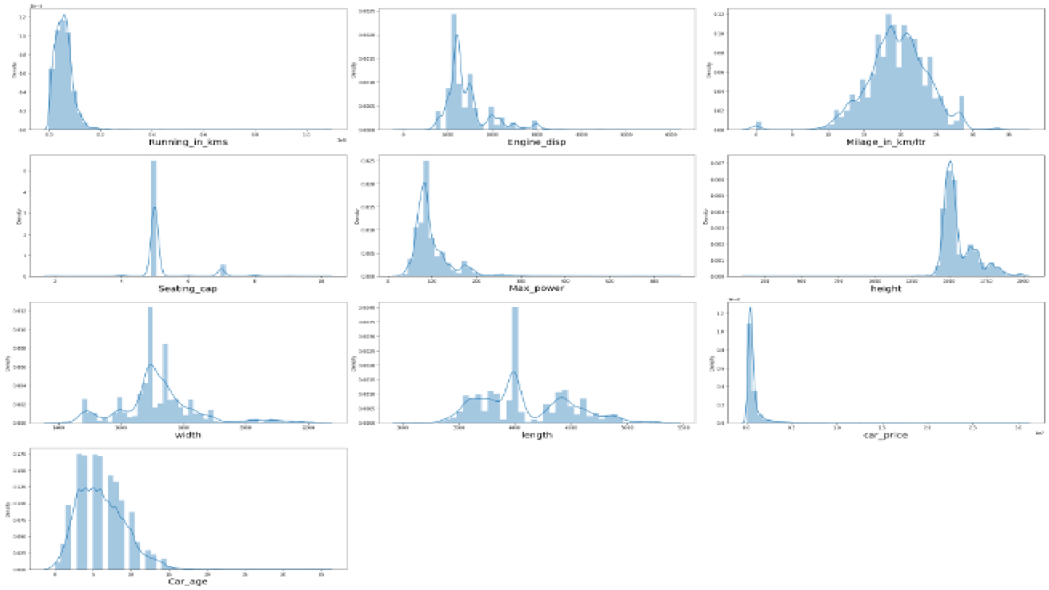
```
# checking for numerical columns
numerical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_columns.append(i)
print(numerical_columns)
```

```
['Running_in_kms', 'Engine_disp', 'Milage_in_km/ltr', 'Seating_cap', 'Max_power', 'height', 'width',
'length', 'car_price', 'Car_age']
```

Above is the list of numerical columns.

Plotting the Distribution plot for all numerical columns

```
# Plotting the Distribution plot for all numerical columns
plt.figure(figsize = (30,20))
plotnumber = 1
for column in df[numerical_columns]:
    if plotnumber <=12:
        ax = plt.subplot(4,3,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize = 20)
    plotnumber+=1
plt.tight_layout()
```

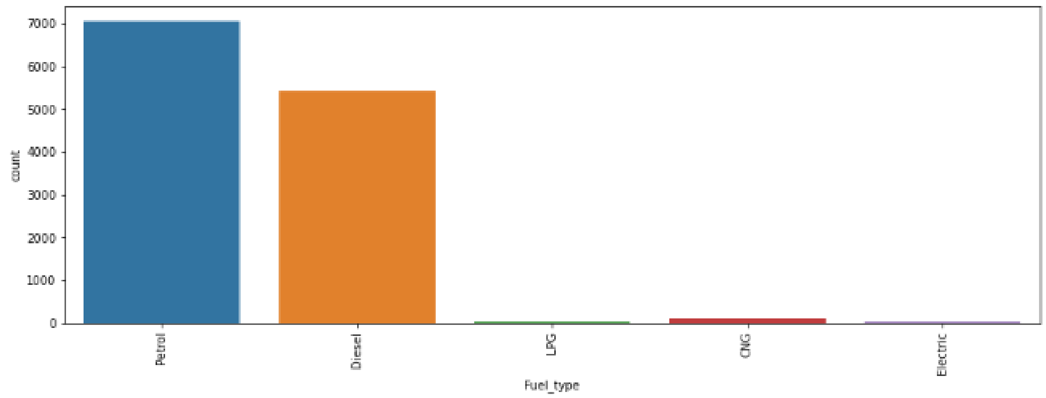


Observations

There is skewness in almost all the numerical columns.

Count plot for Fuel_type column

```
#Count plot for Fuel_type column
plt.figure(figsize=[15,5])
sns.countplot(df['Fuel_type'])
plt.xticks(rotation=90);
```

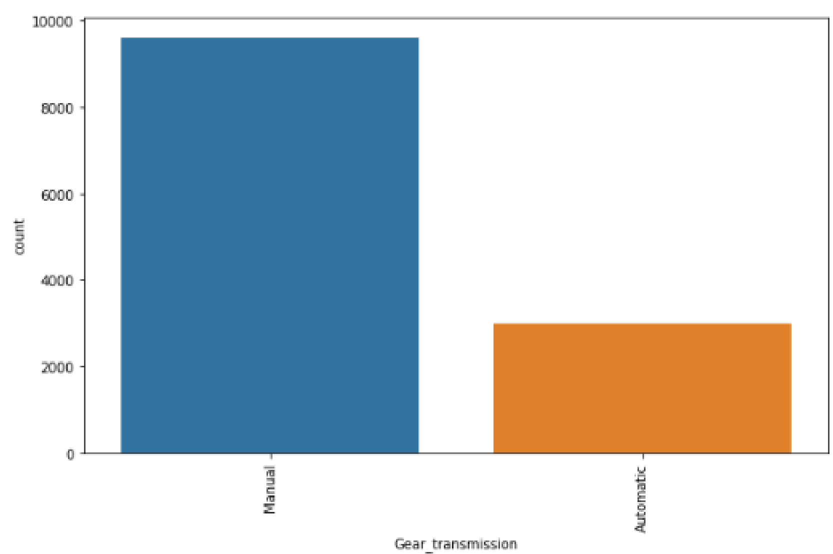


Observations

- Maximum cars are petrol driven and diesel driven cars stand in next position.
- LPG, Electric and CNG are least used fuels for the cars in the given dataset

Count plot for Gear_transmission column

```
#Count plot for Gear_transmission column
plt.figure(figsize=[10,6])
sns.countplot(df['Gear_transmission'])
plt.xticks(rotation=90);
```

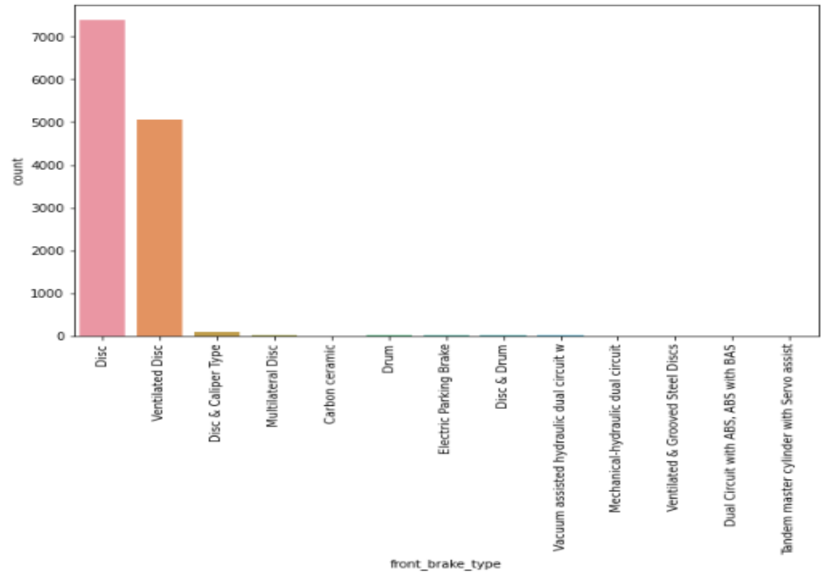


Observations

- Most of the cars have Manual gear transmission.
- Only few of the cars have automatic gear transmission compared to the manual transmission.

Count plot for front_brake_type column

```
#Count plot for front_brake_type column
plt.figure(figsize=[10,6])
sns.countplot(df['front_brake_type'])
plt.xticks(rotation=90);
```

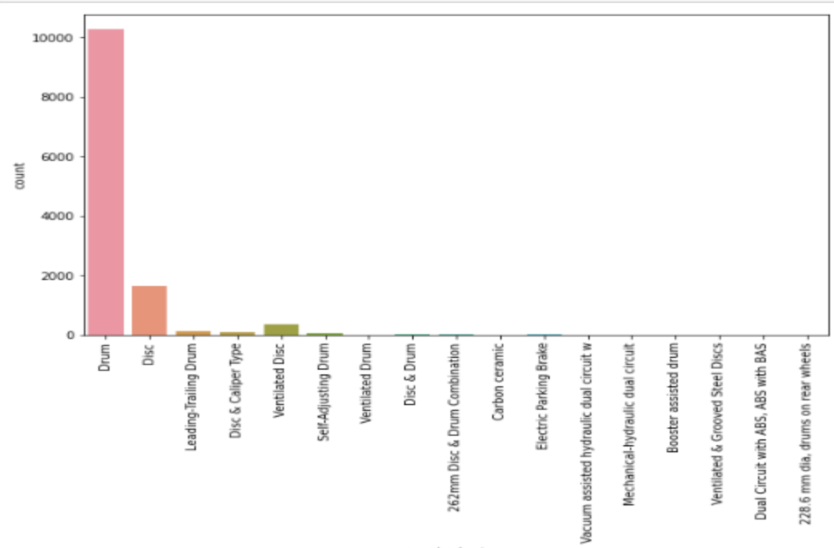


Observations

- Disc front brake cars are more in number followed by Ventilated Disc.

Count plot for rear_brake_type column

```
#Count plot for rear_brake_type column
plt.figure(figsize=[10,6])
sns.countplot(df['rear_brake_type'])
plt.xticks(rotation=90);
```

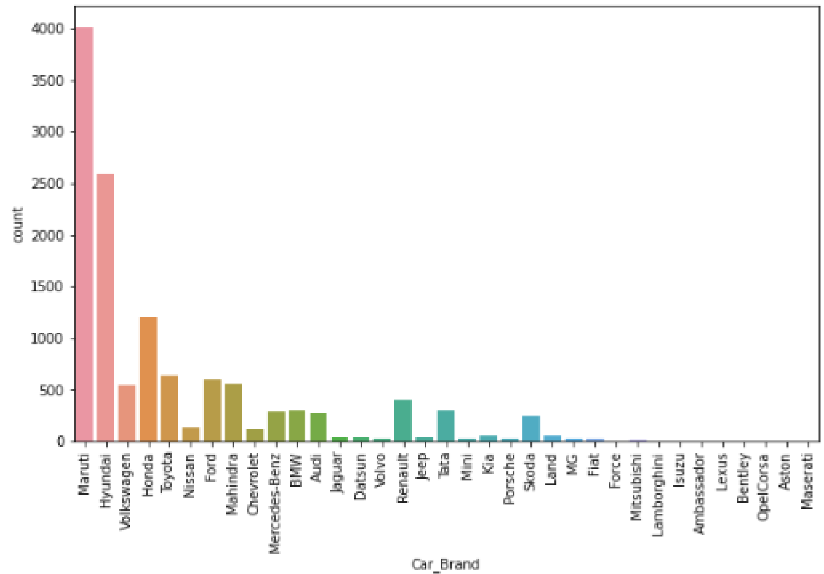


Observations

Drum rare break cars are more in number followed by the Disc rear breaks

Count plot for Car_Brand column

```
#Count plot for Car_Brand column
plt.figure(figsize=[10,6])
sns.countplot(df['Car_Brand'])
plt.xticks(rotation=90);
```

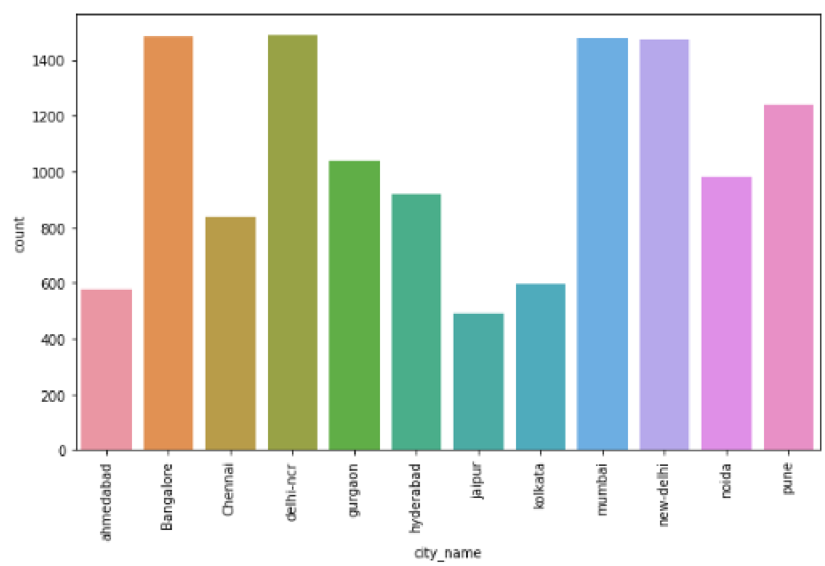


Observations

- Maximum cars under sale in the daatset are Maruti followed by Hyundai.
- Foreign company cars are fewer than the indian made brands

Count plot for city_name column

```
#Count plot for city_name column
plt.figure(figsize=[10,6])
sns.countplot(df['city_name'])
plt.xticks(rotation=90);
```



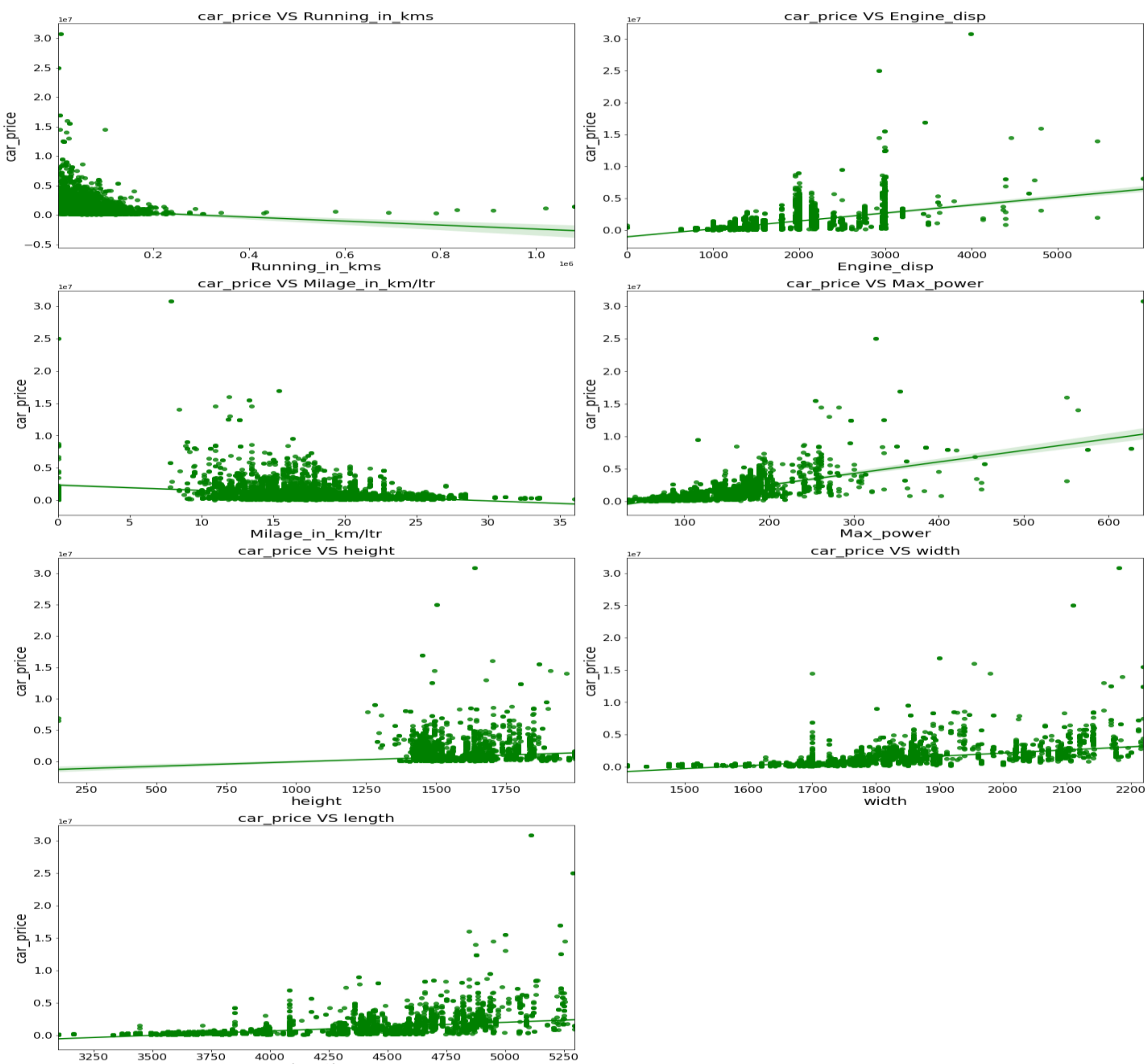
Observations

- In Bangalore,delhi-ncr,mumbai and new-delhi we can find maximum cars for sale. Since these are densly populated places.
- Jaipur and ahmedabad have least number of cars in the dataset

Bivariate Analysis:

```
col=['Running_in_kms', 'Engine_disp', 'Milage_in_km/ltr', 'Max_power', 'height', 'width', 'length']
```

```
#regplot for numerical columns
plt.figure(figsize=(20,40))
for i in range(len(col)):
    plt.subplot(6,2,i+1)
    sns.regplot(x=df[col[i]] , y=df['car_price'],color="g")
    plt.title(f"car_price VS {col[i]}",fontsize=20)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.xlabel(col[i],fontsize = 20)
    plt.ylabel('car_price',fontsize = 20)
    plt.tight_layout()
```

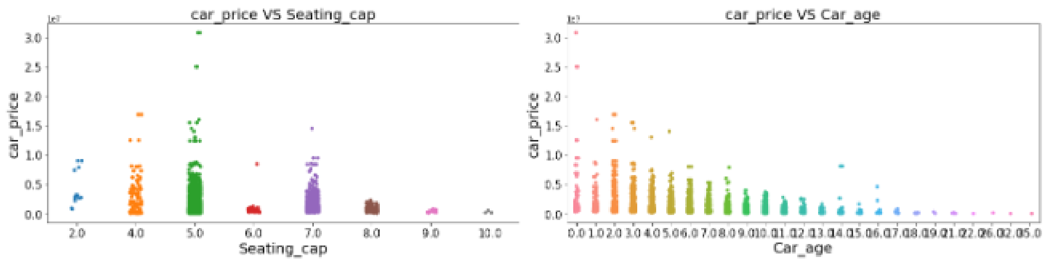


Observations

- Maximum cars are having below 20k driven kms. And car price is high for less driven cars.
- Maximum cars are having 1000-3000 Endine_disp. And car price is high for 3000 Endine_disp.
- Maximum cars are having milage of 10-25kms. And ,milage has no proper relation with car price.
- As Max_power is increasing car price is also increasing.
- Car_price has no proper relation with height.
- As the width is increasing car price is also increasing.
- As length is increasing car price is also increasing.
- Weight also has linear relationship with car price.
- As top_speed is increasing car price is also increasing.

```
col1=['Seating_cap','Car_age']
```

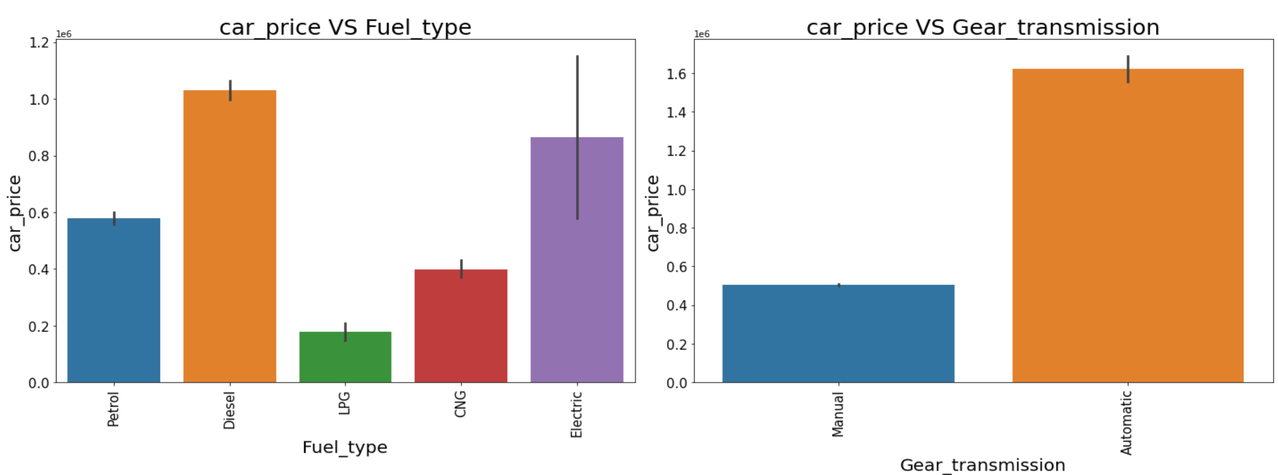
```
#stripplot for numerical columns
plt.figure(figsize=(20,5))
for i in range(len(col1)):
    plt.subplot(1,2,i+1)
    sns.stripplot(x=df[col1[i]] , y=df['car_price'])
    plt.title(f"car_price VS {col1[i]}",fontsize=20)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.xlabel(col1[i],fontsize = 20)
    plt.ylabel('car_price',fontsize = 20)
    plt.tight_layout()
```

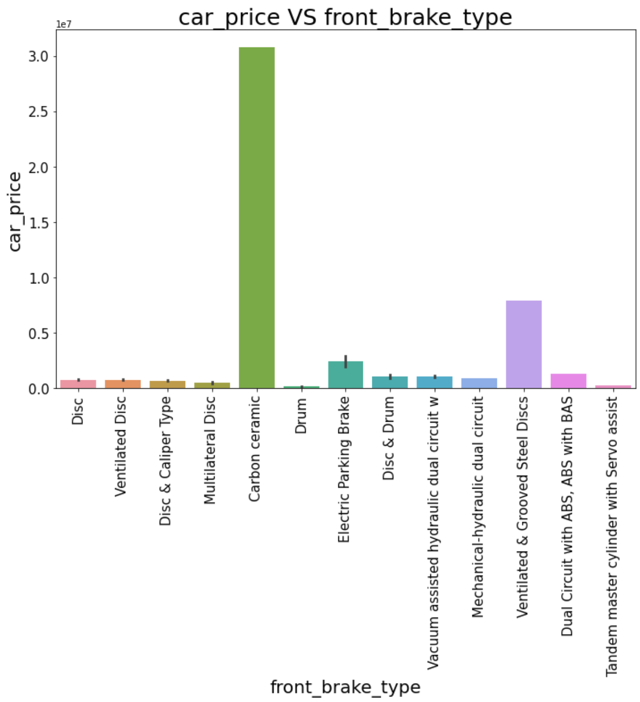


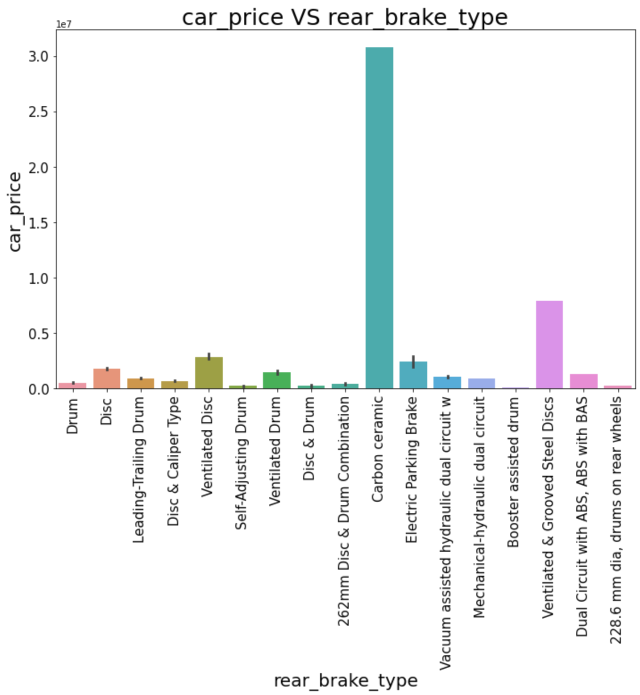
Observations

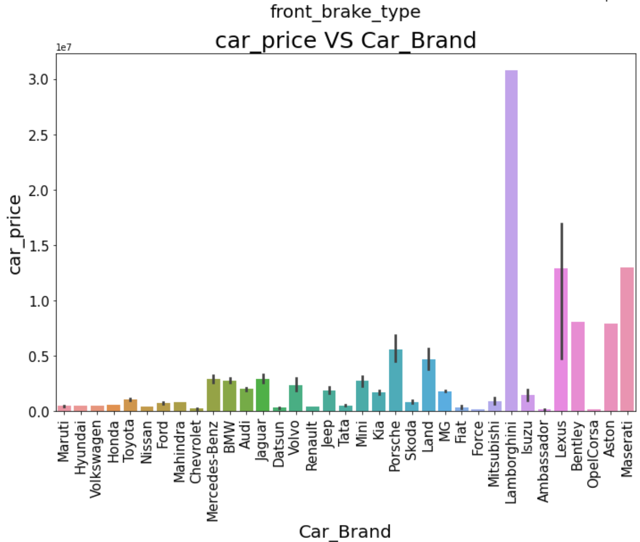
- Cars with 5 and 4 seats are having highest price.
- 5 seater cars have higher prices than the other cars.
- As the age of the car increases the car price decreases. Lesser the age of the car, higher the price.

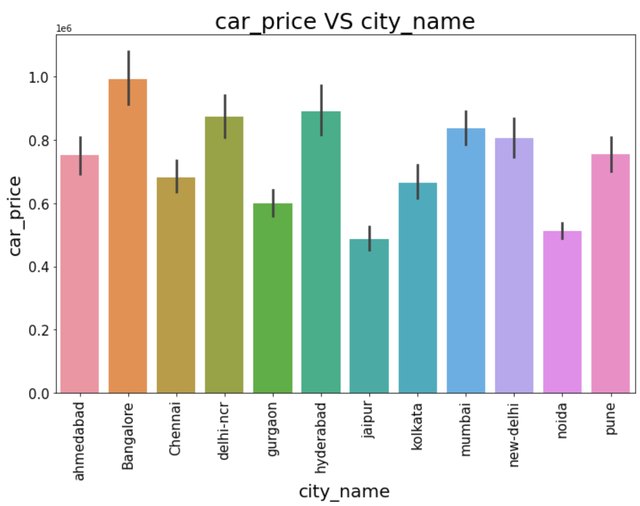
Bar plot for all categorical columns











Observations

- For Diesel and Electric cars the price is high compared to Petrol,LPG and CNG.
- Cars with automatic gear are costlier than manual gear cars.
- Cars with Carbon Ceramic front break are costlier compared to other cars.
- Cars with carbon Ceramic rear braek are costlier compared to other cars.
- Lamborghini brand cars are having highset sale price.
- In Bangalore, Hyderabad and delhi-ncr the car prices are high as they are highly populated cities.

Pair plotting for df

```python
#pair ploting for df
sns.pairplot(df)
```

<seaborn.axisgrid.PairGrid at 0x2b3060d9c70>

Observations

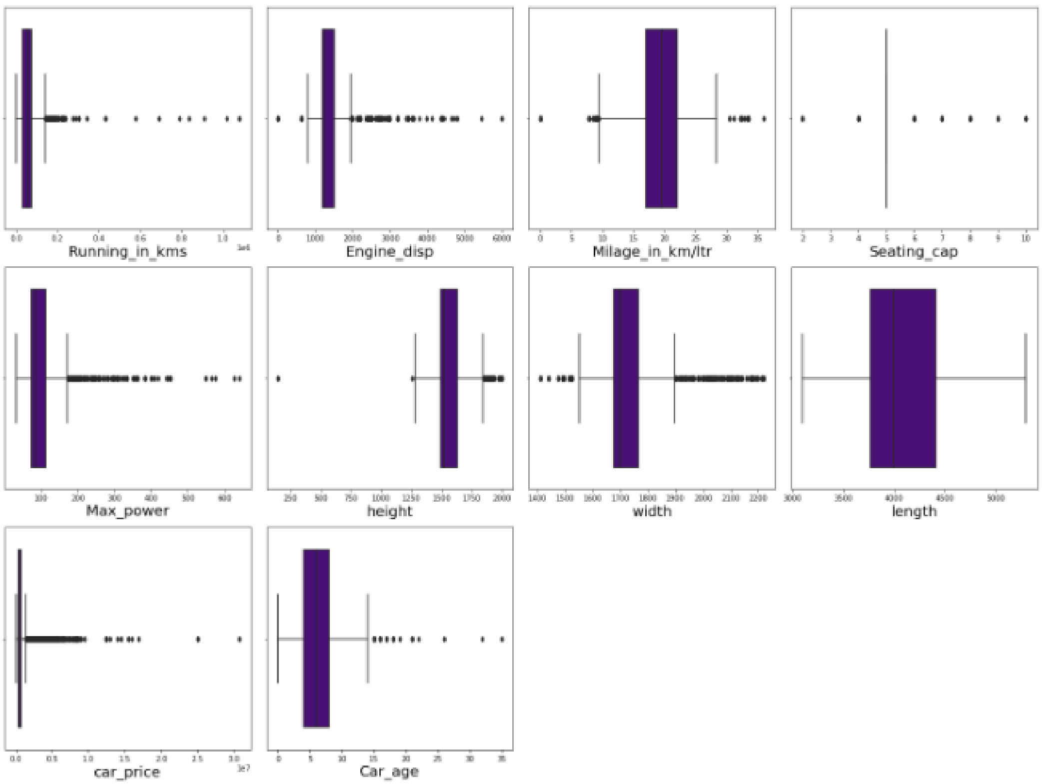
By looking into the pair plot of pair of features we can notice the presence of outliers in each plot so we have to deal with this in the later steps.

- **Identification of possible problem-solving approaches (methods)**

Checking for outliers:

```
# Identifying the outliers using boxplot

plt.figure(figsize=(20,15),facecolor='white')
plotnumber=1
for column in numerical_columns:
    if plotnumber<=30:
        ax=plt.subplot(3,4,plotnumber)
        sns.boxplot(df[column],color='indigo')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Observations

- There are outliers in all columns except length.
- Since car_price is our target we should not remove outliers from it.
- Removing Outliers:

i) Zscore method:

```
#Features having outliers
features=df[['Running_in_kms', 'Engine_disp', 'Milage_in_km/ltr', 'Seating_cap', 'Max_power', 'height',
```

Above are the list of columns with outliers in the dataset.

```
from scipy.stats import zscore
z=np.abs(zscore(features))
df_new=df[(z<3).all(axis=1)]
df_new.head()
```

	Fuel_type	Running_in_kms	Engine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_brake_t
0	Petrol	131125.0	998.0	Manual	21.79	5.0	Grey	67.05	
1	Petrol	73875.0	1197.0	Manual	18.90	5.0	White	82.00	
2	Diesel	97922.0	1498.0	Manual	22.27	5.0	White	108.60	Ventilated
3	Petrol	24230.0	998.0	Manual	21.70	5.0	Red	67.05	Ventilated
4	Petrol	41174.0	998.0	Automatic	20.51	5.0	Grey	67.00	Ventilated

```
#Checking shape of new dataset
df_new.shape
```

```
(11657, 18)
```

The new dataset has 11657 rows and 18 columns.

```
#Checking shape of old dataset
df.shape
```

```
(12608, 18)
```

Previously the dataset has 12608 rows and 18 columns.

Checking dataloss in zscore method

```
#Checking dataloss in zscore method
Dataloss = (((12608-11657)/12608)*100)
Dataloss
```

```
7.542829949238579
```

ii) IQR method:

```
# 1st quantile
Q1=features.quantile(0.25)

# 3rd quantile
Q3=features.quantile(0.75)

# IQR
IQR=Q3 - Q1

df_1=df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

We have removed the skewness of the dataset using IQR method.

```
#Checking shape of new dataset
df_1.shape
```

```
(8725, 18)
```

The new dataset has 8725 rows and 18 columns.

```
#Checking shape of old dataset
df.shape
```

```
(12608, 18)
```

Previously the dataset has 12608 rows and 18 columns.

```
#Checking dataloss in IQR method of the dataset
Dataloss = (((12608-8725)/12608)*100)
Dataloss
```

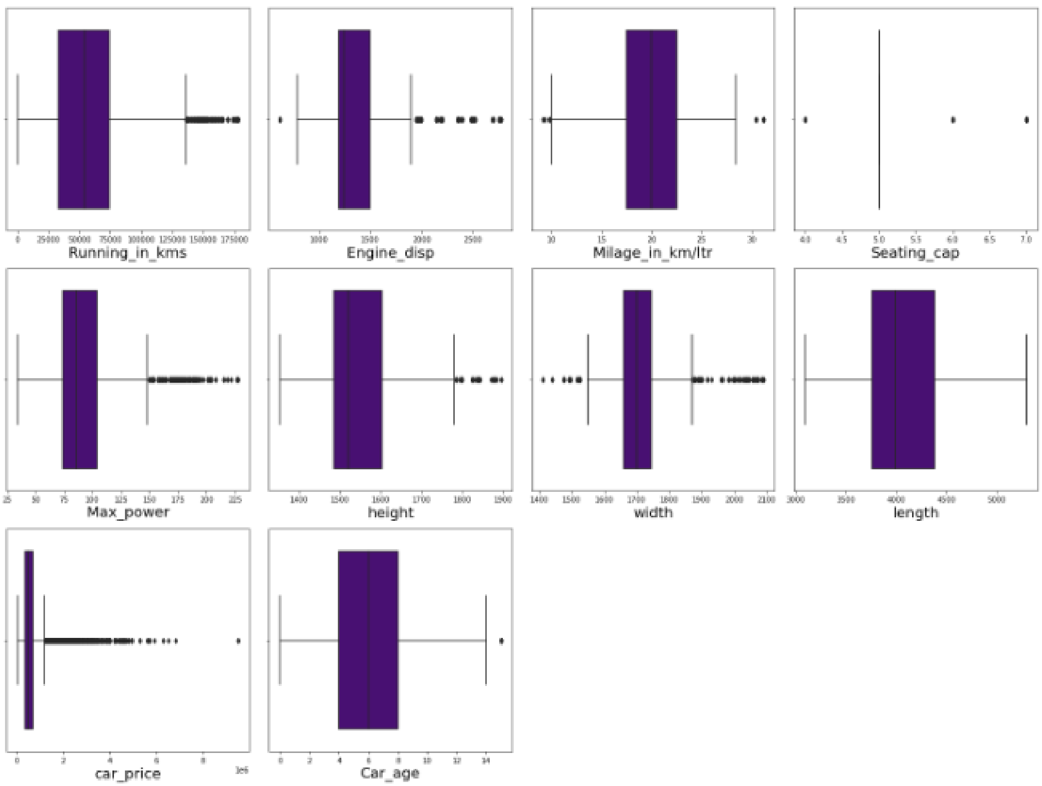
```
30.797906091370557
```

In IQR method the data loss is more than 10%. So let us continue with the dataset obtained after removing the outliers using Z-score method

Checking the outliers again

```
# Checking if the outliers are reduced

plt.figure(figsize=(20,15),facecolor='white')
plotnumber=1
for column in numerical_columns:
    if plotnumber<=30:
        ax=plt.subplot(3,4,plotnumber)
        sns.boxplot(df_new[column],color='indigo')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Observations

Outliers has been reduced in all the columns.

Checking for skewness:

```
# Now checking for numerical columns
num_columns=[]
for i in df_new.dtypes.index:
    if df_new.dtypes[i]!='object':
        num_columns.append(i)
print(num_columns)
```

```
['Running_in_kms', 'Engine_disp', 'Milage_in_km/ltr', 'Seating_cap', 'Max_power', 'height', 'width',
'length', 'car_price', 'Car_age']
```

Checking for skewness in the dataset

```
#Checking for skewness in the dataset
df_new[num_columns].skew()
```

```
Running_in_kms      0.557609
Engine_disp         1.322739
Milage_in_km/ltr    0.079676
Seating_cap         3.028409
Max_power           1.409836
height              1.275817
width               0.381738
length              0.409327
car_price           4.069770
Car_age             0.485301
dtype: float64
```

Observations

We can notice there is skewness in all the numerical columns except Milage_in_km/ltr,width,length and Car_age. So we have to remove this skewness. Since car_price is my target no need to remove skewness in this column.

Removing skewness using yeo-johnson method:

```
#Creating a list of skewed features
fea=['Running_in_kms', 'Engine_disp', 'Seating_cap', 'Max_power', 'height']
```

Taking a list as fea with all the columns with skewness.

```
# Removing skewness
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
'''
parameters:
method = 'box_cox' or 'yeo-johnson'
'''
```

```
"\nparameters:\nmethod = 'box_cox' or 'yeo-johnson'\n"
```

Using yeo_johnson method for removing the skewness.

```
df_new[fea] = scaler.fit_transform(df_new[fea].values)
```

Got removed from skewness.

```
#Checking skewness again
df_new[fea].skew()
```

```
Running_in_kms   -0.067269
Engine_disp      -0.023585
Seating_cap      -1.588158
Max_power        -0.024164
height            0.000000
dtype: float64
```

Observations

In all the columns skewness has reduced and in height column skewness is zero after removing which means this column has single entry throught out. So let us drop this column as it has no impact on model building.

Dropping height column

```
#Droping height column
df_new = df_new.drop(["height"],axis=1)
```

Label Encoding:

Separating categorical columns in df_new

```
# Separating categorical columns in df_new
cat_col=[]
for i in df_new.dtypes.index:
    if df_new.dtypes[i]=='object':
        cat_col.append(i)
print(cat_col)
```

```
['Fuel_type', 'Gear_transmission', 'color', 'front_brake_type', 'rear_brake_type', 'Car_Brand', 'Car_
Model', 'city_name']
```

Above are the list of categorical columns in df_new.

```
from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
df_new[cat_col]= df_new[cat_col].apply(LE.fit_transform)
```

```
df_new[cat_col].head()
```

	Fuel_type	Gear_transmission	color	front_brake_type	rear_brake_type	Car_Brand	Car_Model	city_name
0	4	1	64	0	5	17	186	2
1	4	1	163	0	5	8	89	2
2	1	1	163	6	5	26	178	2
3	4	1	125	6	5	17	142	2
4	4	0	64	6	5	17	186	2

Using label encoder we have encoded the categorical columns.

Correlation

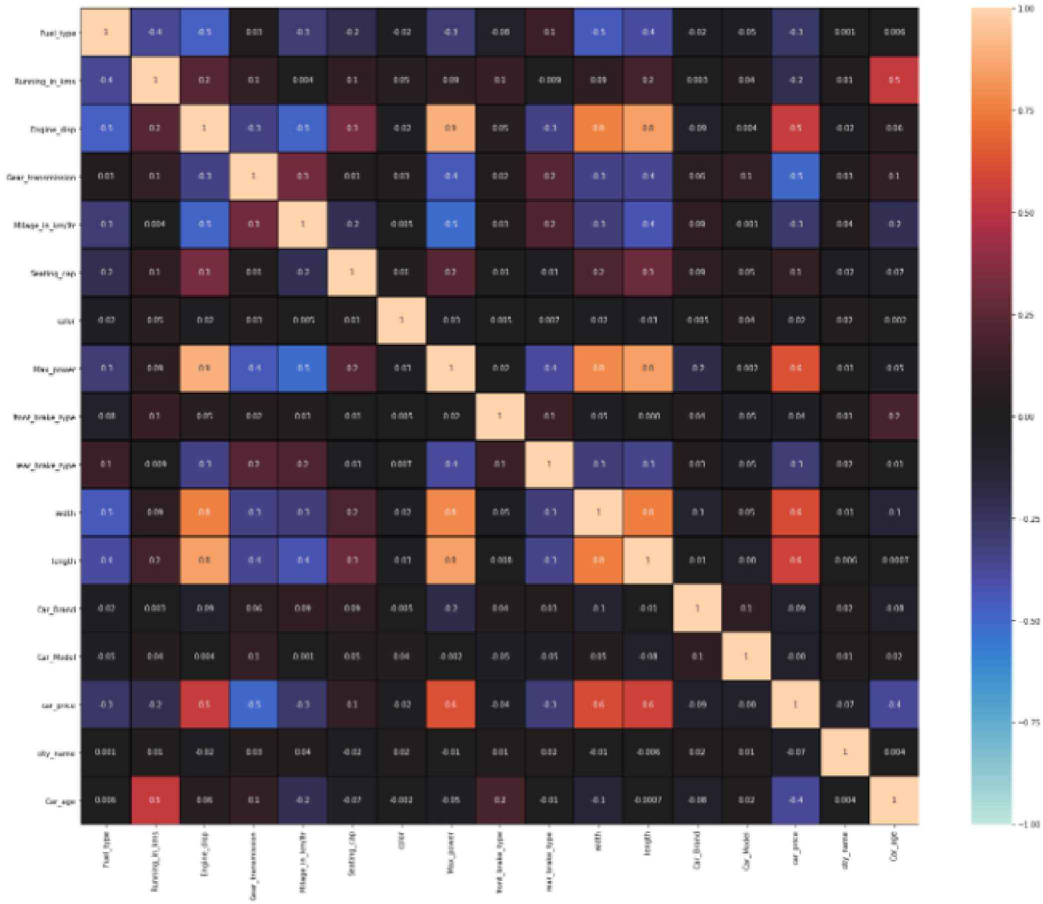
```
# Checking correlation
cor=df_new.corr()
cor
```

	Fuel_type	Running_in_kms	Engine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_
Fuel_type	1.000000	-0.363945	-0.471699	0.025136	-0.296387	-0.209881	-0.022484	-0.
Running_in_kms	-0.363945	1.000000	0.230672	0.112245	0.004170	0.099851	0.047985	0.
Engine_disp	-0.471699	0.230672	1.000000	-0.328223	-0.481663	0.326677	-0.020807	0.
Gear_transmission	0.025136	0.112245	-0.328223	1.000000	0.307897	0.009509	0.025971	-0.
Milage_in_km/ltr	-0.296387	0.004170	-0.481663	0.307897	1.000000	-0.213999	-0.004647	-0.
Seating_cap	-0.209881	0.099851	0.326677	0.009509	-0.213999	1.000000	0.013078	0.
color	-0.022484	0.047985	-0.020807	0.025971	-0.004647	0.013078	1.000000	-0.
Max_power	-0.291857	0.087395	0.894647	-0.439458	-0.513468	0.228783	-0.032854	1.
front_brake_type	-0.079265	0.127323	0.049591	0.023750	0.026566	-0.009978	-0.005270	-0.
rear_brake_type	0.145259	-0.008620	-0.341595	0.229150	0.207954	-0.028265	0.007275	-0.
width	-0.450686	0.093010	0.763400	-0.340643	-0.305911	0.201982	-0.018960	0.
length	-0.388495	0.164753	0.843988	-0.354620	-0.422680	0.291513	-0.032669	0.
Car_Brand	-0.019693	0.003438	-0.092950	0.063166	0.094383	0.085157	-0.004926	-0.
Car_Model	-0.049746	0.039774	0.004066	0.101954	-0.000978	0.046753	0.039580	-0.
car_price	-0.259962	-0.203328	0.540460	-0.495043	-0.279612	0.112843	-0.022555	0.
city_name	0.001462	0.012961	-0.020389	0.027232	0.035684	-0.020176	0.020729	-0.
Car_age	0.005921	0.531894	0.056813	0.116829	-0.226008	-0.067724	-0.002310	-0.

Observations

Above are the correlations of all the pair of features.To get better visualization on the correlation of features, let us plot it using heat map.

```
# Visualizing the correlation matrix by plotting heat map.
plt.figure(figsize=(25,20))
sns.heatmap(df_new.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.1g', annot = True, linecolor="black",an
plt.yticks(rotation=0);
```

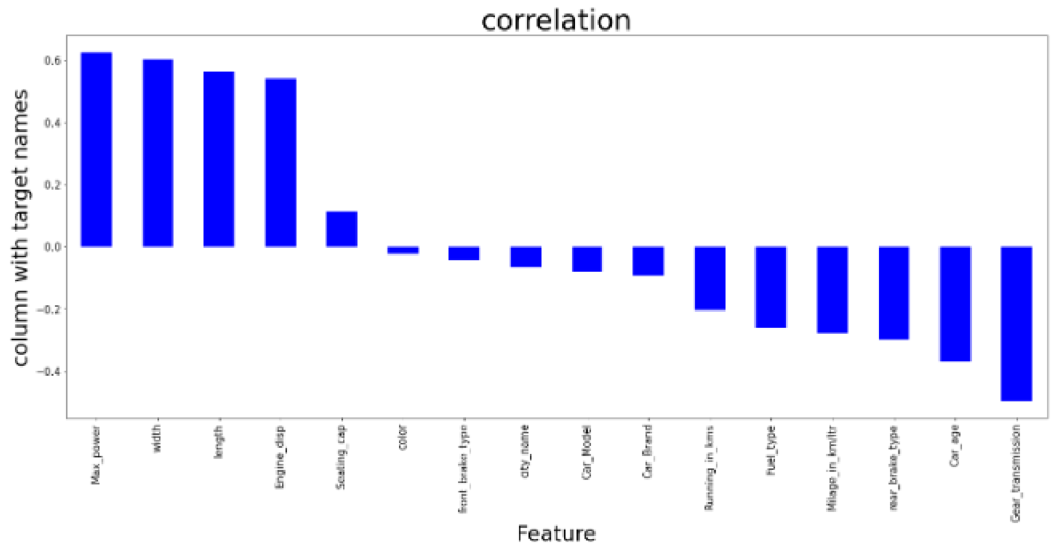


Observations

- Most of the columns in the darker shades are near by zero, which says they donot have much impact on the dataset.
- We can notice there is multicolinearity issue in the dataset. So we have to use VIF to remove multicolinearity.
- Width, lngth, and max power are highly correlated with the engine_disp column which causes multicollinearity in the model
- Our target column 'Car_price' is highly correlated with Width, lngth, and max power
- 'Gear_transmission' is highly negatively correlated with the target column 'Car_price'.
- The columns 'color', 'front_brake_type', 'Car_brand', 'Car_model', 'city_name' have near zero correlation with the target colunn which says there is no impact on the target column due to any change in these columns

Let's visualize the correlation of all the features with target to get better insight.

```python
# visualizing the correlation of all the features with target
plt.figure(figsize=(25,10))
df_new.corr()['car_price'].sort_values(ascending=False).drop(['car_price']).plot(kind='bar',color='b')
plt.xlabel('Feature',fontsize=30)
plt.ylabel('column with target names',fontsize=30)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('correlation',fontsize=40)
plt.show()
```



Observations

Color is less correlated with target. But will keep it and continue.

Separating Features and Target:

```python
# Separating the target and features
x = df_new.drop("car_price",axis=1)
y = df_new["car_price"]
```

We have separated my target and independent columns.

Scaling

```python
# Scaling the data using Standard scaler
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
```

I have scaled my data using Standard scaler.

```python
#Viewing the top 5 columns in X after scaling
X.head()
```

	Fuel_type	Running_in_kms	Engine_disp	Gear_transmission	Milage_in_km/ltr	Seating_cap	color	Max_power	front_bra
0	0.828736	2.122981	-1.162401	0.523307	0.481631	-0.182728	-0.936759	-0.974446	-(
1	0.828736	0.675606	-0.332179	0.523307	-0.318875	-0.182728	1.060064	-0.235563	-(
2	-1.198808	1.324161	0.585258	0.523307	0.614588	-0.182728	1.060064	0.678634	1
3	0.828736	-1.038907	-1.162401	0.523307	0.456702	-0.182728	0.293607	-0.974446	1
4	0.828736	-0.364909	-1.162401	-1.910925	0.127082	-0.182728	-0.936759	-0.977324	1

This is the data of independent variables after scaling.

Checking for multicollinearity

```
# checking the VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Features"]=X.columns
vif
```

	vif_Features	Features
0	3.016718	Fuel_type
1	1.773652	Running_in_kms
2	9.457321	Engine_disp
3	1.368716	Gear_transmission
4	2.867009	Milage_in_km/ltr
5	1.274229	Seating_cap
6	1.011934	color
7	9.589635	Max_power
8	1.097505	front_brake_type
9	1.231269	rear_brake_type
10	3.375083	width
11	4.886744	length
12	1.215606	Car_Brand
13	1.103853	Car_Model
14	1.004444	city_name
15	1.892286	Car_age

Dropping columns with high VIF

```
#Droping columns with high VIF
X = X.drop(["Max_power"],axis=1)
```

```
# Checking the VIF of the dataset again
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Features"]=X.columns
vif
```

	vif_Features	Features
0	2.774992	Fuel_type
1	1.773414	Running_in_kms
2	5.902581	Engine_disp
3	1.302289	Gear_transmission
4	2.866997	Milage_in_km/ltr
5	1.261341	Seating_cap
6	1.011725	color
7	1.097449	front_brake_type
8	1.215609	rear_brake_type
9	3.176195	width
10	4.257093	length
11	1.094394	Car_Brand
12	1.089182	Car_Model
13	1.004218	city_name
14	1.851800	Car_age

We can see that the "Engine_disp" has high VIF, Hence we shall drop this column for better model building

Dropping columns with high VIF

```
#Droping columns with high VIF
X = X.drop(["Engine_disp"],axis=1)
```

```
# Checking the VIF of the dataset again
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Features"]=X.columns
vif
```

	vif_Features	Features
0	2.318471	Fuel_type
1	1.763102	Running_in_kms
2	1.302241	Gear_transmission
3	2.305033	Milage_in_km/ltr
4	1.254257	Seating_cap
5	1.010636	color
6	1.080208	front_brake_type
7	1.209721	rear_brake_type
8	2.974388	width
9	3.121165	length
10	1.079093	Car_Brand
11	1.086601	Car_Model
12	1.004154	city_name
13	1.849374	Car_age

Now we can see that the multicollinearity issue is solved.

Finding Best Random State and Accuracy:

```
#importing necessary libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

```
# Finding Best Random State and Accuracy
from sklearn.ensemble import RandomForestRegressor
maxAccu=0
maxRS=0
for i in range(1,200):
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30, random_state =i)
    mod = RandomForestRegressor()
    mod.fit(X_train, y_train)
    pred = mod.predict(X_test)
    acc=r2_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.9633628934196624  on Random_state  8
```

We have got the best accuracy and random state.

Creating Train test split

```
# Train test splitting the data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.30,random_state=maxRS)
```

Created train test split.

Regression Algorithms:

```
#importing necessary libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.linear_model import SGDRegressor
from xgboost import XGBRegressor
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingRegressor
from sklearn import metrics
```

i) RandomForestRegressor:

```
RFR=RandomForestRegressor()
RFR.fit(X_train,y_train)
pred=RFR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(RFR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 96.24309718113597
mean_squared_error: 13343319258.574087
mean_absolute_error: 57808.42825605562
root_mean_squared_error: 115513.28606949977

Cross validation score : 93.01825638123728

R2_Score - Cross Validation Score : 3.2248407998986863
```

RFR is giving 96.24% r2_score.

ii) XGBRegressor:

```
XGB=XGBRegressor()
XGB.fit(X_train,y_train)
pred=XGB.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(XGB, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 97.10845957621967
mean_squared_error: 10269828335.682026
mean_absolute_error: 55197.8550416086
root_mean_squared_error: 101340.1615139922

Cross validation score : 92.90590742501206

R2_Score - Cross Validation Score : 4.2025521512076125
```

XGBRegressor is giving 97.10% r2_score.

iii) GradientBoostingRegressor:

```
GBR=GradientBoostingRegressor()
GBR.fit(X_train,y_train)
pred=GBR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(GBR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 93.12768539007554
mean_squared_error: 24408267210.199486
mean_absolute_error: 87750.42444333046
root_mean_squared_error: 156231.45397198186

Cross validation score : 89.74508273759753

R2_Score - Cross Validation Score : 3.3826026524780133
```

GradientBoostingRegressor is giving 93.12% r2_score.

iv) DecisionTreeRegressor:

```
DTR=DecisionTreeRegressor()
DTR.fit(X_train,y_train)
pred=DTR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(DTR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 93.17091264475825
mean_squared_error: 24254737803.74087
mean_absolute_error: 66098.62550028588
root_mean_squared_error: 155739.32645205857

Cross validation score : 87.54990624741913

R2_Score - Cross Validation Score : 5.621006397339116
```

DecisionTreeRegressor is giving 93.17% r2_score.

v) Bagging Regressor:

```python
BR=BaggingRegressor()
BR.fit(X_train,y_train)
pred=BR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(BR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 94.17486869218727
mean_squared_error: 20689006479.74771
mean_absolute_error: 64832.85456995834
root_mean_squared_error: 143836.73550156687

Cross validation score : 92.46857232597438

R2_Score - Cross Validation Score : 1.7062963662128965
```

Bagging Regressor is giving 94.17% r2_score.

The XGBReggressor has 97.10 as r2_score but has a higher difference between the r2_score and the Cross validation score than the Random forest regressor

By evaluating based on the difference of model accuracy i.e., r2_score and cross validation score we can say that RandomForestRegressor as the best model with 96.24% r2_score.

▪ Testing of Identified Approaches (Algorithms)

Hyper parameter tuning for best model:

```python
#importing necessary libraries
from sklearn.model_selection import GridSearchCV
```

```python
parameter = {'n_estimators':[30,60,80],
             'max_depth': [10,20,40],
             'min_samples_leaf':[1,2,5,10,20,30],
             'min_samples_split':[5,10,20],
             'criterion':['mse','mae'],
             'max_features':["auto","sqrt","log2"]}
```

Giving RandomForestRegressor parameters.

```python
GCV=GridSearchCV(RandomForestRegressor(),parameter,cv=5,n_jobs = -1,verbose = 1)
```

Running grid search CV for RandomForestRegressor.

```python
#Running grid search CV for RFR
GCV.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 972 candidates, totalling 4860 fits

GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'criterion': ['mse', 'mae'], 'max_depth': [10, 20, 40],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 5, 10, 20, 30],
                         'min_samples_split': [5, 10, 20],
                         'n_estimators': [30, 60, 80]},
             verbose=1)
```

▪ Run and Evaluate selected models

Tuning the model using GCV.

Obtaining the best parameters

```
#Getting the best parameters
GCV.best_params_
```

```
{'criterion': 'mae',
 'max_depth': 40,
 'max_features': 'log2',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 80}
```

Got the best parameters for RandomForestRegressor.

```
Best_model=RandomForestRegressor(criterion='mae',max_features='log2',min_samples_split=5,n_estimators=8
Best_model.fit(X_train,y_train)
pred=Best_model.predict(X_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print("RMSE value:",np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
R2_Score: 94.61499485464296
mean_squared_error: 19125818880.05539
mean_absolute_error: 66146.4632647227
RMSE value: 138296.12749479065
```

This is the final model with 94.61% as r2_score after tuning which is good.

```
# Saving the model using .pkl
import joblib
joblib.dump(Best_model,"Used_Car_Price.pkl")
```

```
['Used_Car_Price.pkl']
```

Saving the model:

```
# Loading the saved model
model=joblib.load("Used_Car_Price.pkl")

#Prediction
prediction = model.predict(X_test)
prediction
```

```
array([232950.  , 271400.  , 300725.  , ..., 232987.5 , 282168.75,
       400862.5 ])
```

We have saved the model as Used_Car_Price Using .pkl

Predictions:

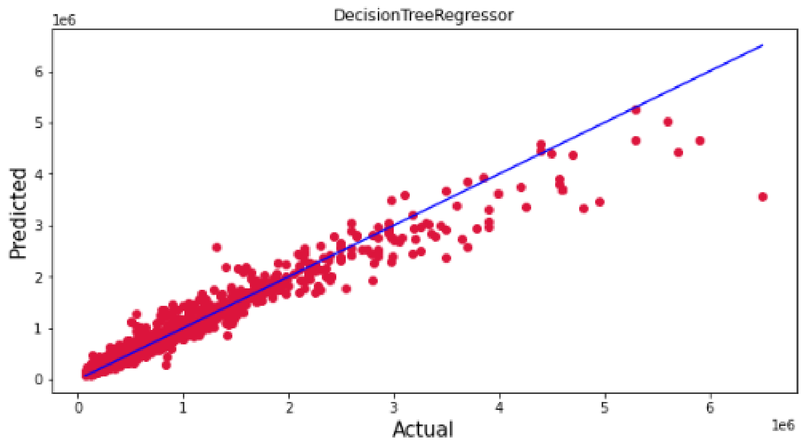
```
#Creating a dataaframe for the predicted vs actual values
pd.DataFrame([model.predict(X_test)[:],y_test[:]],index=["Predicted","Actual"])
```

	0	1	2	3	4	5	6	7	8	9	10
Predicted	232950.0	271400.0	300725.0	349631.25	319975.0	428150.0	541562.5	826418.75	1832068.75	674406.25	452718.75
Actual	350000.0	230000.0	296000.0	342000.00	330000.0	475000.0	575000.0	789000.00	2275000.00	755000.00	446000.00

Above are the predicted values and the actual values, which look similarwith a few exceptions

Plotting a graph for the actual values vs predicted values

```
# plotting a graph for the actual values vs predicted values
plt.figure(figsize=(10,5))
plt.scatter(y_test, prediction, c='crimson')
p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('Actual', fontsize=15)
plt.ylabel('Predicted', fontsize=15)
plt.title("DecisionTreeRegressor")
plt.show()
```



Observation

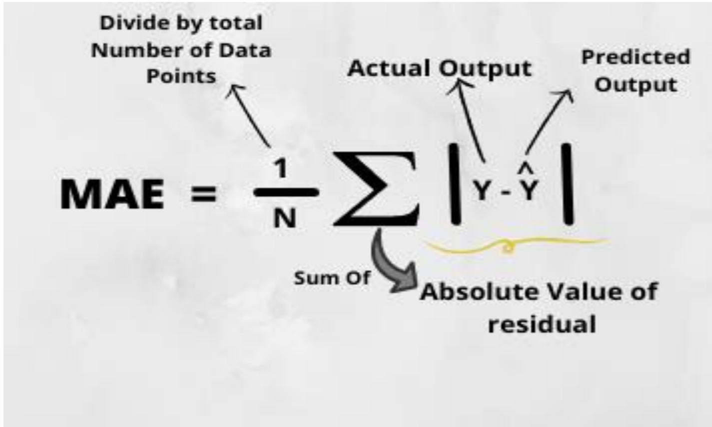
We have plotted the Actual vs Predicted. To get better insights - Bule line is the actual line and red dots are the predicted values

We can observe that the predicted values are nearby the actual values which says the model built works well.

- **Key Metrics for success in solving problem under consideration**

The essential step in any machine learning model is to evaluate the accuracy and determine the metrics error of the model. I have used Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R2 Score metrics for my model evaluation:

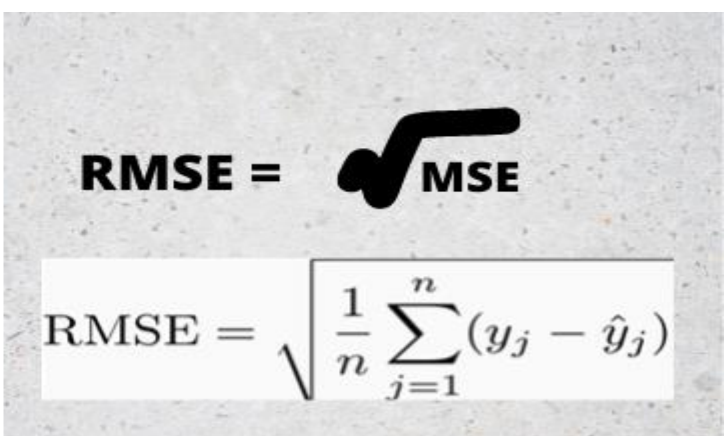
- **Mean Absolute Error (MAE):** MAE is a popular error metric for regression problems which gives magnitude of absolute difference between actual and predicted values. The MAE can be calculated as follows:



- **Mean Squared Error (MSE):** MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value. We perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

$$MSE = \frac{1}{n}\,\Sigma\,\underbrace{\left(y - \widehat{y}\right)}_{\text{The square of the difference between actual and predicted}}^{2}$$

- **Root Mean Squared Error (RMSE):** RMSE is an extension of the mean squared error. The square root of the error is calculated, which means that the units of the RMSE are the same as the original units of the target value that is being predicted.



- **R2 Score:** I have used R2 score which gives the accurate value for the models used. On the basis of R2 score I have created final model.

▪ Interpretation of the Results

The dataset should be cleaned and scaled to build the ML models to get good predictions. I have performed few processing steps which I have already mentioned in the pre-processing steps where all the important features are present in the dataset and ready for model building.

In univariate analysis I have used count plots and pie plots to visualize the counts in categorical variables and distribution plot to visualize the numerical variables. In bivariate analysis I have used reg plots, bar plots, strip plots, line plots and violin plot to check the relation between label and the features. Used pair plot to check the pairwise relation between the features. The heat map and bar plot helped me to understand the correlation between dependent and independent features. Also, heat map helped to detect the multicollinearity problem and feature importance. Detected outliers and skewness with the help of box plots and distribution plots respectively. And I found some of the features skewed to right as well as to left. I got to know the count of each column using bar plots.

After cleaning and processing data, I performed train test split to build the model. I have built multiple regression models to get the accurate R2 score, and evaluation metrics like MAE, MSE and RMSE. I got Extreme Gradient Boosting Regressor (XGB) as the best model which gives 96.27% R2 score. I checked the cross-validation score ensuring there will be no overfitting and underfitting. After tuning the best model, the R2 score of Extreme Gradient Boosting Regressor has been increased to 96.90% and also got low evaluation metrics. Finally, I saved my final model and got the good predictions results for price of used cars.

Conclusion

▪ Key Findings and Conclusions of the Study

The case study aims to give an idea of applying Machine Learning algorithms to predict the sale price of the used cars. After the completion of this project, we got an insight of how to collect data, pre-processing the data, analyzing the data, cleaning the data and building a model.

In this study, we have used multiple machine learning models to predict the sale price of the used cars. We have gone through the data analysis by performing feature engineering, finding the relation between features and label through visualizations. And got the important feature and we used these features to predict the car price by building ML models. After training the model we checked CV score to overcome with the overfitting issue. Performed hyper parameter tuning on the best model and the best model's R2 score increased and was giving R2 score as 96.90%. We have also got good prediction results of car price.

From the whole study we got to know that the continuous numerical variables having some strong positive linear relation with the label "Car_Price". By comparing car price and categorical variables we got to know that the cars having automatic gear transmission, cars from the city Bangalore, cars using petrol and diesel as fuels, cars having the brands Benz and BMW and cars with 5-7 seating capacity have high sale price. While comparing continuous numerical variables and Car_Price we found that cars which are having good mileage, engine displacement, less running in kms have good linear relation with the price that is the cars with this kind of qualities have high selling prices. We found outliers and removed them and further removed skewness. Looking at the heat map, I could see there were some features which were correlated with each other, so I used VIF method to remove the feature causing multicollinearity and scaled the data to overcome with the data biasness.

▪ Learning Outcomes of the Study in respect of Data Science

While working on this project I learned many things about the features of cars and about the car selling web platforms and got the idea that how the machine learning models have helped to predict the price of used cars which provides greater understanding into the many causes and benefits of selling old cars. I found that the project was quite interesting as the dataset contains several types of data. I used several types of plotting to visualize the relation between target and features. This graphical representation helped me to understand which features are important and how these features describe selling price of the old cars. Data cleaning was one of the important and crucial things in this project where I dealt with features having string values, features extraction and selection. Finally got XGBoosting Regressor as best model.

The challenges I faced while working on this project was when I was scrapping the real time data from cardekho website, it took so much time to gather data. The data was quite difficult to handle and cleaning part was challenging for me but fixed it well and it was unable to remove skewness in Seating_cap column so moved further keeping it as it is.

Finally, our aim was achieved by predicting the sale price of used cars and built car price evaluation model that could help the clients to understand the future price of used cars.

▪ Limitations of this work and Scope for Future Work

The main limitation of this study is the low number of records that have been used. In the dataset our data is not properly distributed in some of the columns many of the values in the columns are “-“and some values which are not realistic. Because I have seen the column running in kms showing 0 kms and some of the cars having age as 0 years which are not possible in case of used cars. So, because of that data our models may not make the right patterns and the performance of the model also reduces. So that issues need to be taken care.

Future work: As future work, we intend to collect more data and to use more advanced techniques like artificial neural networks and genetic algorithms to predict car prices. In future this machine learning model may bind with various website which can provide real time data for price prediction. Also, we may add large historical data of car price which can help to improve accuracy of the machine learning model.