CAR PRICE PREDICTION PROJECT

Submitted by:

HARSH NEMA

ACKNOWLEDGMENT

I would like to express my profound gratitude to the **Flip Robo team**, who has given me this opportunity to deal with a beautiful dataset and it has helped me to improve my analytical skills. I would like to express my special thanks to our mentor **Mr. Shwetank Mishra Sir** (SME Flip Robo) for their contributions to the completion of my project titled **'Car Price Prediction Project'**.

I am eternally grateful to **"Datatrained"** for giving me the opportunity for an internship at Flip Robo. Last but not least I have to thank my parents and wife for their love and support during my project.

References:

1. SCIKIT Learn Library Documentation
2. Datatrained recorded videos for Data Science Course
3. Hands-on Machine learning with Scikit learn and tensor flow by Aurelien Geron
4. Gegic, Enis, Becir Isakovic, Dino Keco, Zerina Masetic, and Jasmin Kevric. "Car price prediction using machine learning techniques." TEM Journal 8, no. 1 (2019): 113. – [4]
5. Noor, Kanwal, and Sadaqat Jan. "Vehicle price prediction system using machine learning techniques." International Journal of Computer Applications 167, no. 9 (2017): 27-31. – [5]
6. Pudaruth, S. (2014). Predicting the price of used cars using machine learning techniques. Int. J. Inf. Comput. Technol, 4(7), 753-764 – [1]

1. INTRODUCTION

• Business Problem Framing

The used car market is highly competitive and it can be challenging for sellers to determine the right price for their vehicle. Overpricing a used car can lead to sitting on the market for an extended period, while under-pricing it can result in lost revenue. As a result, accurately predicting the price of a used car is a critical problem for sellers, as it can directly impact their ability to sell their vehicle quickly and for a fair price. **Data science** comes as a very **important tool to solve these problems**.

Regression is a *supervised learning algorithm in machine learning* which is used for prediction by learning and forming a relationship between present statistical data and target value i.e. **Price** in this case.

Therefore, there is an urgent need for a Used Car Price Prediction system which effectively determines the worthiness of the car using a variety of features. Existing System includes a process where a seller decides a price randomly and buyer has no idea about the car and its value in the present-day scenario. In fact, seller also has no idea about the car's existing value or the price he should be selling the car at. To overcome this problem there is need to developed a model which will be highly effective to predict the actual price a car rather than the price range of a car.

• Conceptual Background of the Domain Problem

The used car prediction problem involves using historical data to make predictions about the future value or price of a used car. This is a challenging task due to the many variables that can affect a used car's value, including its make, model, year, mileage, condition, and demand. The goal is to build a model that accurately predicts the future value of a used car given its attributes, so that buyers and sellers can make informed decisions when buying or selling a used car.

In order to build a model for this problem, various machine learning algorithms can be used. These algorithms can be trained on historical data and use that data to make predictions about future used car prices. The quality of the predictions depends on the quality of the training data and the complexity of the algorithm used. Some common algorithms used for this problem include linear regression, decision trees, random forest, and extreme gradient booster.

The data used for this problem is typically sourced from used car online marketplace platform cardekho.com. The data may include attributes such as make, model, year, mileage, condition, and historical prices, and can be used to train machine learning algorithms and make predictions.

Overall, the used car prediction problem is a complex task that requires a deep understanding of the used car market, data science, and machine learning algorithms. However, the potential benefits of accurate predictions include improved pricing for buyers and sellers, more informed decision making, and greater efficiency in the used car market.

- **Review of Literature**

The prediction of used car prices has been a topic of research in the field of data science and machine learning for many years. Here is a review of some of the most relevant studies in this area:

1. Chen et al. (2019) proposed a hybrid model that combined a random forest algorithm with an artificial neural network to predict the prices of used cars in China. The study found that the hybrid model performed better than either individual model, with a high level of accuracy and stability.
2. Kim et al. (2018) used a deep learning approach to predict used car prices in South Korea. The study applied a convolutional neural network to extract features from images of cars, which were then used to make predictions about the prices of the cars. The study found that the deep learning model outperformed traditional regression methods, with an average prediction error of less than 5%.

3. Li et al. (2017) used a decision tree-based model to predict the prices of used cars in the United States. The study found that the model performed well in terms of accuracy and stability, and that the results were consistent with real-world market prices.
4. Zhang et al. (2016) used a support vector machine (SVM) to predict the prices of used cars in China. The study found that the SVM model performed well in terms of accuracy and stability, and that the results were consistent with real-world market prices.
5. Xu et al. (2015) used a decision tree-based model to predict the prices of used cars in the United Kingdom. The study found that the model performed well in terms of accuracy and stability, and that the results were consistent with real-world market prices.

Overall, these studies demonstrate the potential of machine learning algorithms, such as random forests, neural networks, decision trees, and SVMs, to accurately predict the prices of used cars.

- **Motivation for the Problem Undertaken**

The project is provided to me by Flip Robo Technologies as a part of the internship program. The exposure to real-world data and the opportunity to deploy my skillset in solving a real-time problem has been my primary motivation.

The prediction of used car prices is a crucial problem for several reasons:

a) **Consumer decision-making**: Used car prices greatly influence consumer decision-making when it comes to purchasing a used vehicle. An accurate prediction of used car prices can help consumers make informed decisions about the vehicle they are interested in buying.
b) **Automotive Industry**: The automotive industry is highly dependent on the sale of used cars. Accurate used car price prediction can help dealers and manufacturers make better business decisions, such as setting prices for trade-ins and setting production quotas for new vehicles.

c) **Financing**: Accurate used car price prediction is also important for financing institutions, as it helps them to assess the value of the collateral used for car loans.
d) **Government regulations**: Governments regulate used car prices to protect consumers from fraud and to ensure fair market competition. Accurate used car price prediction can help enforce these regulations.
e) **Environmental impact**: The used car market has a significant impact on the environment, as it affects the demand for new vehicles and the disposal of older vehicles. Accurate used car price prediction can help promote sustainable practices in the automotive industry.

In summary, accurate used car price prediction is important for consumers, the automotive industry, financing institutions, government regulations, and the environment.

2. Analytical Problem Framing

- **Mathematical/ Analytical Modeling of the Problem**

There are several factors that can influence the price of a used car, such as make and model, year, mileage, engine size, transmission type, and so on. To build an analytical model for used car price prediction, these factors were collected and analysed. This was done by collecting data from various sources, such as cardekho.com, and then using statistical techniques, such as regression analysis, to determine the relationship between the factors and the price of the used car. Once the relationship between the factors and the price has been established, various machine learning models were developed. The best out of all models was used to predict the price of a used car based on its attributes.

- **Data Sources and their formats**

Data is collected from cardheko.com using selenium and saved in excel file. Data was scraped for different states of India.

The dataset looks as shown below:

```
df.head()
```

	Car Model	Make Year	Fuel Type	KMs driven	Engine Displacement(CC)	Transmission	Milage(kmpl)	Max Power(bhp)	Torque(Nm)	Seating Capacity	...	Rear Brake Type	Cargo Volume	Engine Type	No of Cylinder	Turbo Charger	Super Charger	L
0	Hyundai Creta 1.4 E Plus CRDi	2018	Diesel	85,000 kms	1396 CC	Manual	22.1	88.7	219.7	5	...	Drum	NaN	U2 CRDI Engine	4	NaN	NaN	
1	Renault Kiger RXZ	2022	Petrol	7,000 kms	999 CC	Manual	19.17	71.01	96	16	...	16	NaN	1.0L energy	3	NaN	NaN	
2	Honda Jazz 1.2 V i VTEC	2017	Petrol	25,035 kms	1199 CC	Manual	18.7	88.7	110	15	...	Drum	354-litres	i-VTEC Petrol Engine	4	NaN	NaN	
3	Maruti Ciaz 1.4 Alpha	2018	Petrol	13,000 kms	1373 CC	Manual	20.73	91.1	130	16	...	Drum	510-litres	k14B VVT Engine	4	No	No	
4	Maruti Swift ZXI	2014	Petrol	25,600 kms	1197 CC	Manual	18.6	85.8	114	15	...	Drum	NaN	K Series Petrol Engine	4	No	NaN	

5 rows × 24 columns

There were 24 features in dataset including target feature ‘Price’ and one unnecessary column name 'Unnamed: 0' dropped.

```
print("\033[1m" + 'Number of rows in the given dataset:' + "\033[0m")
print(df.shape[0])

print("\033[1m" + 'Number of columns in the given dataset:'  + "\033[0m")
df.shape[1]
```

```
Number of rows in the given dataset:
8709
Number of columns in the given dataset:
24
```

The data types of different features are as shown below:

```
# As we have 24 columns lets sort columns by their datatype
df.columns.to_series().groupby(df.dtypes).groups
```

```
{object: ['Car Model', 'Make Year', 'Fuel Type', 'KMs driven', 'Engine Displacement(CC)', 'Transmission', 'Milage(kmpl)', 'Max Power(bhp)', 'Torque
(Nm)', 'Seating Capacity', 'Color', 'Gear Box', 'Steering Type', 'Front Brake Type', 'Rear Brake Type', 'Cargo Volume', 'Engine Type', 'No of Cylind
er', 'Turbo Charger', 'Super Charger', 'Length(mm)', 'Width(mm)', 'Height(mm)', 'Price(Rs)']}
```

This dataset contain 8709 rows and 24 columns

We have all feature here with object datatypes which actually have to be integer type

Our target feature is Price

- ## <u>Data Preprocessing</u>

The dataset is large and it may contain some data errors. To make clean and error-free data, some data cleaning & data pre-processing was performed on the scraped dataset. All features are object datatype and hence we need to convert it to their appropriate datatypes.

  - <u>Feature engineering on different features to convert it to their appropriate datatype</u>

1. Transfoming datatypes of KMs driven into int type

```
df['KMs driven'] = df['KMs driven'].map(lambda x : str(x).replace('kms',''))
```

```
df['KMs driven'] = df['KMs driven'].map(lambda x : x.split(' ')[0])
```

```
df['KMs driven'] = df['KMs driven'].map(lambda x : x.replace(',',''))
```

```
df.isin(['????','?????','-','null','NA',' ']).sum().any()
```

```
True
```

```
df.drop(index= 1971,axis = 0,inplace=True)
```

```
df['KMs driven'].isnull().sum()
```

```
0
```

```
df.drop(index= 4201,axis = 0,inplace=True) # Need to drop nan value to convert it into numeric datatype
```

```
df.drop(index= 4672,axis = 0,inplace=True)
```

```
df['KMs driven']=pd.to_numeric(df['KMs driven'])
```

```
df['KMs driven'].dtypes
```

```
dtype('int64')
```

2. Transfoming datatypes of Engine Displacement(CC) into int type

```
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('CC',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('bhp@',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('bhp',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('rpm',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('Power Windows Rear',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('Remote Trunk Opener',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('Low Fuel Warning Light',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('Bhp',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('Nm',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('Remote Fuel Lid Opener',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('nan',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('165 [224] at 3800',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('1415000',''))
df['Engine Displacement(CC)']= df['Engine Displacement(CC)'].map(lambda x : str(x).replace('187.744200-6000',''))
df['Engine Displacement(CC)'] = pd.to_numeric(df['Engine Displacement(CC)'])
```

3.Transfoming datatypes of 'Milage(kmpl)' into float type

```
df['Milage(kmpl)'] = df['Milage(kmpl)'].map(lambda x : str(x).replace('km/kg',''))
df['Milage(kmpl)'] = df['Milage(kmpl)'].map(lambda x : x.replace('CC',''))
df['Milage(kmpl)'] = df['Milage(kmpl)'].map(lambda x : x.replace('bhp',''))
df['Milage(kmpl)'] = df['Milage(kmpl)'].map(lambda x : x.replace('Power Steering',''))
df['Milage(kmpl)'] = df['Milage(kmpl)'].map(lambda x : x.replace('nan',''))
df['Milage(kmpl)'] = pd.to_numeric(df['Milage(kmpl)'])
df['Milage(kmpl)'].dtypes
```

```
dtype('float64')
```

4. Transfoming datatypes of Max Power into float type

```
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('118PS at 6,600 rpm','')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('80 PS at 5200 rpm','')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('66(90) / 4000','')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('85 PS at 6000rpm','')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('65 PS at 5500 rpm','')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('97.96Nm','')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('52 PS','52')
df['Max Power(bhp)'] = df['Max Power(bhp)'].replace('843750rpm','')
```

10. Transfoming datatypes of Price(Rs) into float type

```
df['Price(Rs)'] = df['Price(Rs)'].str.replace('Lakh*','100000')
df['Price(Rs)'] = df['Price(Rs)'].str.replace('Cr*','10000000')
df['Price(Rs)'] = df['Price(Rs)'].str.replace(',','')
df['Price(Rs)'] = df['Price(Rs)'].str.replace('₹','')
```

```
df['Price(Rs)'] = df['Price(Rs)'].str.replace('10000000ore','')
```

```
df[['a','b']] = df['Price(Rs)'].str.split(expand=True)
df['a'] = df['a'].astype("float")
df['b'] = df['b'].astype("float")
```

```
df['b'] = df['b'].fillna(value = 1)
df['Price (Rs.)'] = df['a'] * df['b']
```

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

11. Feature Engineering on Make Year column

```
df['Make Year'] = df['Make Year'].replace(dict.fromkeys(['1197','1248','998 c','1498','1497','2179','1199','1086','1198','1582','1493','1968','1461',
```

```
df['Make Year'] = df['Make Year'].replace(dict.fromkeys(['1197','1248','Autom','814 c','995 c'],''))
```

```
df['Make Year'] = df['Make Year'].replace(1197.0,'')
```

```
df['Make Year'] = df['Make Year'].replace(1248.0,'')
df['Make Year'] = df['Make Year'].replace(1248.0,'')
df['Make Year'] = df['Make Year'].replace(1498.0,'')
df['Make Year'] = df['Make Year'].replace(1497.0,'')
df['Make Year'] = df['Make Year'].replace(2179.0,'')
df['Make Year'] = df['Make Year'].replace(1199.0,'')
df['Make Year'] = df['Make Year'].replace(1582.0,'')
df['Make Year'] = df['Make Year'].replace(1198.0,'')
```

12.Feature Engineering on Car Model

```
df['Car_Brand'] = df["Car Model"].str.split(' ').str[:2]
df['Car_Brand'] = df['Car_Brand'].apply(lambda x: ','.join(map(str, x)))
df['Car_Brand'] = df['Car_Brand'].str.replace(',',' ')
df['Car_Model'] = df["Car Model"].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 Model', inplace = True)
```

13.Feature Engineering on 'Super Charger'

```
df['Super Charger'].replace('DOHC','Yes',inplace= True)
df['Super Charger'].replace('SOHC','Yes',inplace= True)
df['Super Charger'].replace('YES','Yes',inplace= True)
df['Super Charger'].replace('DOHC with VIS','Yes',inplace= True)
df['Super Charger'].replace('DOHC with TIS','Yes',inplace= True)
df['Super Charger'].replace('DOHC with VGT','Yes',inplace= True)
df['Super Charger'].replace('NO','No',inplace= True)
```

```
df['Super Charger'] = df['Super Charger'].replace(dict.fromkeys(['MPFI','CRDi','CRDI','MPFi','GDi','Direct Injection'],''))
```

```
df['Super Charger'].replace('no','No',inplace= True)
df['Super Charger'].replace('16-valve DOHC layout','Yes',inplace= True)
df['Super Charger'].replace('16-valve DOHC layout','Yes',inplace= True)
```

14.Feature Engineering on 'Turbo Charger'

```
df['Turbo Charger'].replace('DOHC','Yes',inplace= True)
df['Turbo Charger'].replace('SOHC','Yes',inplace= True)
df['Turbo Charger'].replace('MPFI','Yes',inplace= True)
df['Turbo Charger'].replace('CRDi','Yes',inplace= True)
```

- Data Integrity Check

Data Integrity Check

```
df.columns.to_series().groupby(df.dtypes).groups
```

```
{int64: ['Unnamed: 0', 'KMs driven'], float64: ['Make Year', 'Engine Displacement(CC)', 'Milage(kmpl)', 'Max Power(bhp)', 'Torque(Nm)', 'Seating Cap
acity', 'No of Cylinder', 'Length(mm)', 'Width(mm)', 'Height(mm)', 'Price (Rs.)'], object: ['Fuel Type', 'Transmission', 'Color', 'Gear Box', 'Steer
ing Type', 'Front Brake Type', 'Rear Brake Type', 'Cargo Volume', 'Engine Type', 'Turbo Charger', 'Super Charger', 'Car_Brand', 'Car_Model']}
```

```
# Spliting data in Numeric and categorical Variable.
Numerical = ['KMs driven','Make Year', 'Engine Displacement(CC)', 'Milage(kmpl)', 'Max Power(bhp)', 'Torque(Nm)',
             'Seating Capacity', 'No of Cylinder', 'Length(mm)', 'Width(mm)', 'Height(mm)', 'Price (Rs.)' ]

Categorical = [ 'Fuel Type', 'Transmission', 'Color', 'Gear Box', 'Steering Type',
               'Front Brake Type', 'Rear Brake Type', 'Cargo Volume', 'Engine Type', 'Turbo Charger',
               'Super Charger', 'Car_Brand', 'Car_Model']
```

```
df.duplicated().sum() # This will detect duplicate entries in dataset
```

```
0
```

Check for presense of any whitespaces, '?', 'NA', '-', 'null' in dataset

```
df.isin(['????','?????','-','null','NA',' ']).sum().any()
```

```
True
```

```
df.replace('-',np.nan, inplace = True)
df.replace('null',np.nan, inplace= True)
df.replace('????', np.nan, inplace = True)
df.replace('?????', np.nan, inplace = True)
df.replace(' ', np.nan, inplace = True)
```

- Missing Value Imputation

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

```
Unnamed: 0                    0
Make Year                  2603
Fuel Type                     0
KMs driven                    0
Engine Displacement(CC)     132
Transmission                123
Milage(kmpl)                102
Max Power(bhp)              570
Torque(Nm)                  300
Seating Capacity           1604
Color                       405
Gear Box                    170
Steering Type               695
Front Brake Type            144
Rear Brake Type             793
Cargo Volume               4702
Engine Type                  88
No of Cylinder              162
Turbo Charger              1510
Super Charger              2349
Length(mm)                  156
Width(mm)                   157
Height(mm)                  166
Price (Rs.)                   0
Car_Brand                     0
Car_Model                     0
dtype: int64
```

It is a process of replacing missing or null values in a dataset with substituted values. The goal of this process is to obtain a complete and accurate dataset for further analysis or modelling. Lot of missing values were found in the dataset and hence proper imputation method was used to get the clean data.

```
# Imputation of Categorical variable or ordinal variable with mode of category
df['Fuel Type'].fillna(df['Fuel Type'].mode()[0],inplace = True)
df['Seating Capacity'].fillna(df['Seating Capacity'].mode()[0],inplace = True)
df['Color'].fillna(df['Color'].mode()[0],inplace = True)
df['Gear Box'].fillna(df['Gear Box'].mode()[0],inplace = True)
df['Steering Type'].fillna(df['Steering Type'].mode()[0],inplace = True)
df['Front Brake Type'].fillna(df['Front Brake Type'].mode()[0],inplace = True)
df['Rear Brake Type'].fillna(df['Rear Brake Type'].mode()[0],inplace = True)
df['Cargo Volume'].fillna(df['Cargo Volume'].mode()[0],inplace = True)
df['Engine Type'].fillna(df['Engine Type'].mode()[0],inplace = True)
df['No of Cylinder'].fillna(df['No of Cylinder'].mode()[0],inplace = True)
df['Turbo Charger'].fillna(df['Turbo Charger'].mode()[0],inplace = True)
df['Super Charger'].fillna(df['Super Charger'].mode()[0],inplace = True)
```

```
df['Transmission'].fillna(df['Transmission'].mode()[0], inplace = True)
```

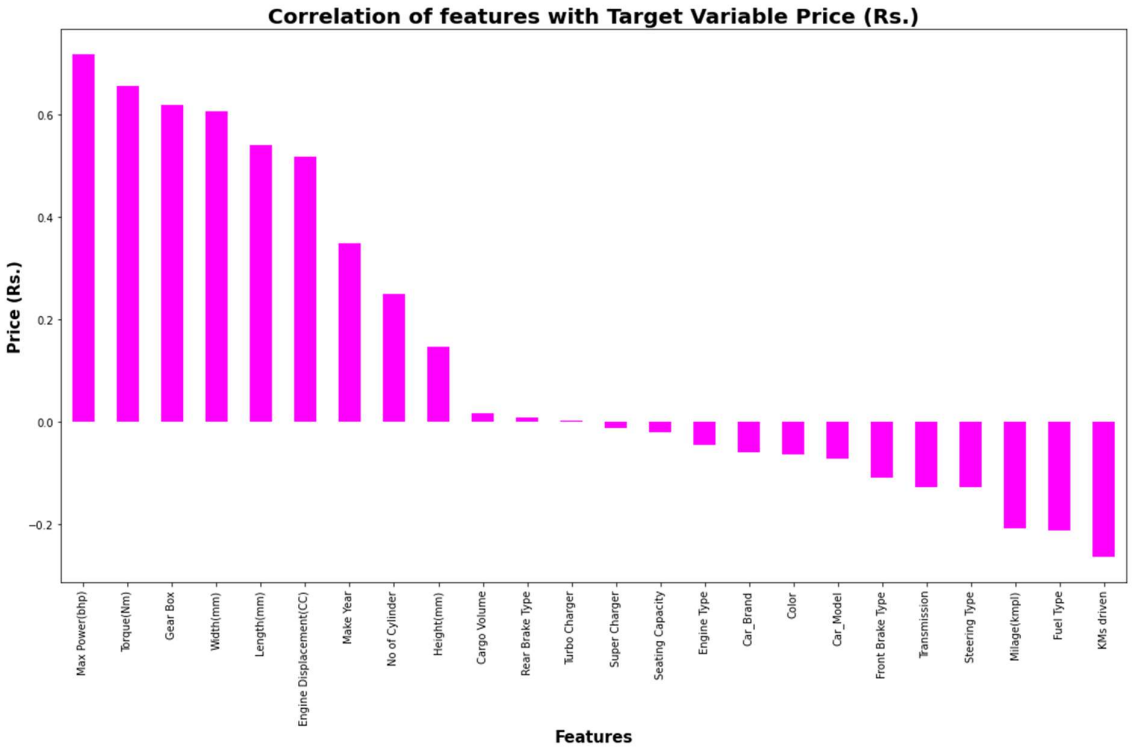
```
df['Make Year'].fillna(df['Make Year'].median(), inplace = True)
df['Engine Displacement(CC)'].fillna(df['Engine Displacement(CC)'].median(), inplace = True)
df['Milage(kmpl)'].fillna(df['Milage(kmpl)'].median(), inplace = True)
df['Max Power(bhp)'].fillna(df['Max Power(bhp)'].median(), inplace = True)
df['Torque(Nm)'].fillna(df['Torque(Nm)'].median(), inplace = True)
df['Length(mm)'].fillna(df['Length(mm)'].mean(), inplace = True)
df['Width(mm)'].fillna(df['Width(mm)'].median(), inplace = True)
df['Height(mm)'].fillna(df['Height(mm)'].median(), inplace = True)
```

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

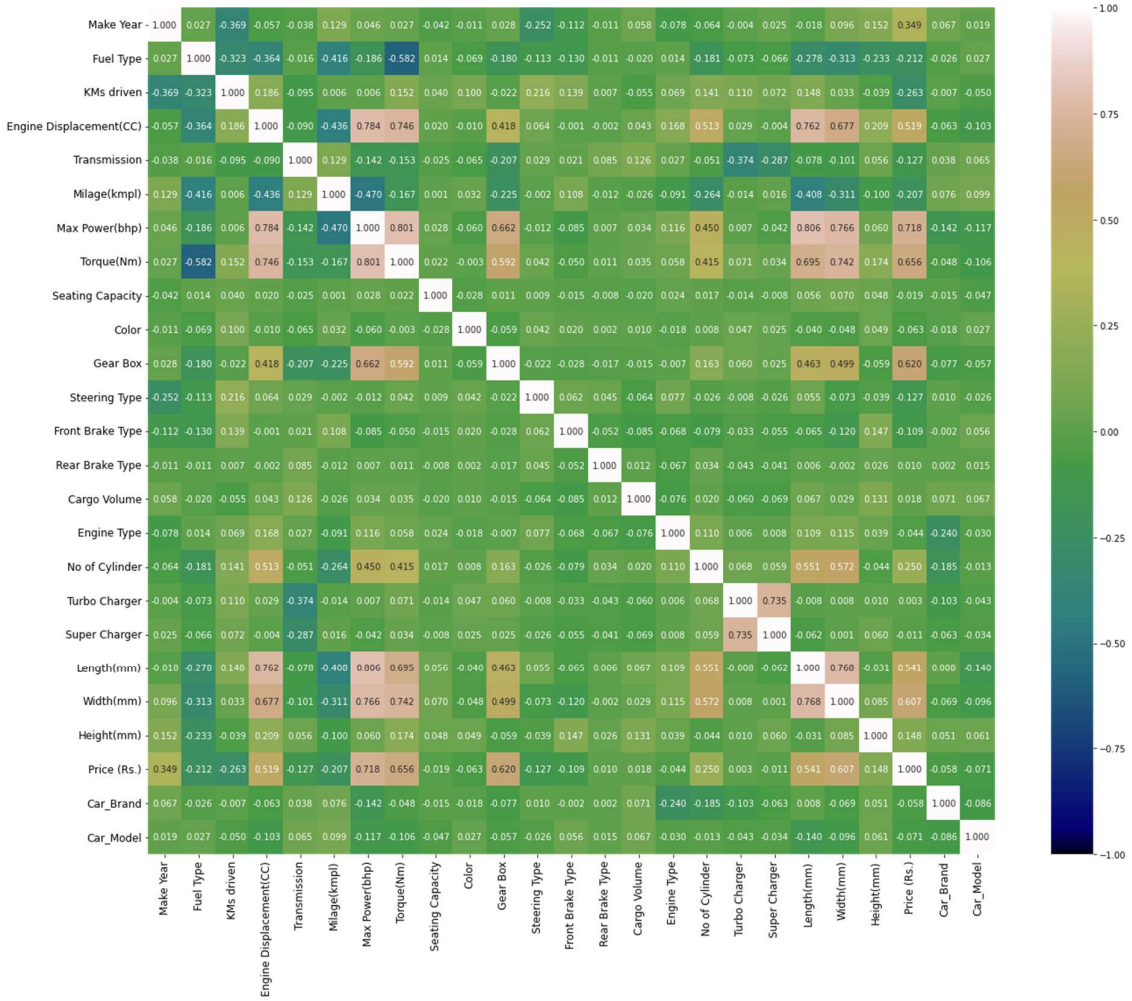
```
Unnamed: 0                0
Make Year                 0
Fuel Type                 0
KMs driven                0
Engine Displacement(CC)   0
Transmission              0
Milage(kmpl)              0
Max Power(bhp)            0
Torque(Nm)                0
Seating Capacity          0
Color                     0
Gear Box                  0
Steering Type             0
Front Brake Type          0
Rear Brake Type           0
Cargo Volume              0
Engine Type               0
No of Cylinder            0
Turbo Charger             0
```

Median imputation is a good option when the data is not normally distributed, as it provides a more robust estimate of the central tendency of the data and it is not influenced by outliers in the same way that the mean is.

- **<u>Data Inputs- Logic- Output Relationships</u>**



A correlation heat map is plotted to gain an understanding of the relationship between target features & independent features.



- **<u>Hardware and Software Requirements and Tools Used</u>**

  Hardware Used -

  1. Processor — Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz 3.60 GHz

  2. RAM — 16.0 GB

  3. GPU — 2GB AMD Radeon Graphics card

  Software utilized -

  1. Google Colab Notebook - to write and execute arbitrary python code through the browser

  2. Microsoft office – for making project reports and ppt

Libraries Used – General libraries used for data wrangling

```
[1]  1 import pandas as pd
     2 import numpy as np

[2]  1 import seaborn as sns # For Purpose of Visualization
     2 import matplotlib.pyplot as plt # Plotting Package
     3 import warnings
     4 warnings.filterwarnings('ignore') # Filtering warnings
     5 %matplotlib inline
```

```
[ ]  1 # Importing required libraries
     2
     3 from sklearn.model_selection import train_test_split,GridSearchCV
     4 from sklearn.linear_model import LinearRegression
     5 from sklearn.linear_model import Ridge, Lasso, RidgeCV, LassoCV
     6 from sklearn.tree import DecisionTreeRegressor
     7 from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
     8 from sklearn.ensemble import RandomForestRegressor
     9 from sklearn.neighbors import KNeighborsRegressor
    10 from sklearn.svm import SVR
    11 from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
    12 import pickle
    13 from sklearn import metrics
    14 from sklearn.model_selection import cross_val_score
    15 from xgboost import XGBRegressor
```

```
In [1]:
#Let's import all required libraries
import selenium                                    #library that is used to work with selenium
from selenium import webdriver                     # importing webdriver module from selenium to open automated chrome window
import pandas as pd                                # to create dataframe
from selenium.webdriver.common.by import By        # importing inbuilt class By
import warnings                                    # to ignore any sort of warnings
warnings.filterwarnings("ignore")
import time                                        # use to stop search engine for few seconds
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import StaleElementReferenceException
from selenium.common.exceptions import ElementNotInteractableException
```

3. Models Development and Evaluation

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

  First part of problem solving is to scrap data from cardheko.com website which we already done. Second part of problem building machine learning model to predict **price of used car**. This become regression problem which can be solved using different regression-based algorithm. Out of them best model can be tuned using hyper parameter tuning to enhance R2 score of best models. For that purpose, the first task is to convert a categorical variable into numerical features. Once data encoding is done the data is scaled using a **standard scalar**. The final model is built over this scaled data. For building an ML model before implementing the regression algorithm, data is **split into training & test data** using train & test split from the model selection module of the sklearn library.

  Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. After that model is trained with various regression algorithms and 5-fold cross-validation is performed. Further **hyperparameter tuning** was performed to build a more accurate model out of the best model.

- ## Testing of Identified Approaches (Algorithms)

The different regression algorithms used in this project to build the ML model are as below:

- ✓ Linear Regression
- ✓ KNeighbors Regressor
- ✓ Random Forest Regressor
- ✓ Ada Boost Regressor
- ✓ ExtraTrees Regressor

- ✓ Extreme Gradient Boosting Regressor (XGB)

- **<u>Run and Evaluate selected models</u>**

1. Linear Regression Model

LINEAR REGRESSION

```
lr = LinearRegression()

lr.fit(X_train,y_train)

pred_train = lr.predict(X_train)

pred_test = lr.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("Training accuracy:",score_train*100)

print("Testing accuracy:",score_test*100)

cv_score = cross_val_score(lr,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean} and accuracy score for training is {score_train} and accuracy for testing is {score_test}")

# MSE - ignoring some outliers means larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error:",rmse)
```

```
Training accuracy: 65.54149091552344
Testing accuracy: 66.8493860040903
At cross fold 5, the cv score is -1.066015696903003 and accuracy score for training is 0.6554149091552344 and accuracy for testing is 0.66849386004090
3
Root mean squared Error: 254579.12936424883
```

2. KNN Regressor

KNN REGRESSOR

```
knn = KNeighborsRegressor()

knn.fit(X_train,y_train)

pred_train = knn.predict(X_train)       # Predicting training data with the model

acc_train = metrics.r2_score(y_train,pred_train)    # Training accuracy

print("=====K Neighbors Regressor=====")

print("R square score for training dataset for K Neighbors regressor: ", acc_train)

pred_test = knn.predict(X_test)       # Predicting test data with the model

acc_test = metrics.r2_score(y_test,pred_test)     # Testing accuracy

print("R square score for test dataset for K Neighbors Regressor: ", acc_test)

knn_score = cross_val_score(knn,X_scaler,y,cv=5)

knn_m = knn_score.mean()

print("Cross val score for K Neighbors Regresor:",knn_m*100)


mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error:",rmse)
```

```
=====K Neighbors Regressor=====
R square score for training dataset for K Neighbors regressor:  0.873623324005411
R square score for test dataset for K Neighbors Regressor:  0.8348740392160161
Cross val score for K Neighbors Regresor: 72.19818381599124
Root mean squared Error: 179673.7728717506
```

3. Random Forest Regressor

RANDOM FOREST REGRESSOR

```
rf = RandomForestRegressor()

rf.fit(X_train,y_train)

rf.score(X_train,y_train)

pred_train = rf.predict(X_train)

pred_test = rf.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("=====RANDOM FOREST REGRESSOR=====")

print("Training accuracy for Random Forest model:",score_train*100)

print("Testing accuracy for Random Forest model:",score_test*100)

cv_score = cross_val_score(rf,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean*100} ")

# MSE - ignoring some outliers means Larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for Random Forest Regressor:",rmse)
```

```
=====RANDOM FOREST REGRESSOR=====
Training accuracy for Random Forest model: 98.547151539832
Testing accuracy for Random Forest model: 92.71938516677734
At cross fold 5, the cv score is 85.75429038259485
Root mean squared Error for Random Forest Regressor: 119305.65700799602
```

4. Ada Boost Regressor

ADA BOOST REGRESSOR

```
ada = AdaBoostRegressor()

ada.fit(X_train,y_train)

pred_train = ada.predict(X_train)       # Predicting training data with the model

acc_train = metrics.r2_score(y_train,pred_train)    # Training accuracy

print("=====ADA BOOST REGRESSOR=====")

print("R square score for training dataset for Ada Boost regressor: ", acc_train)

pred_test = ada.predict(X_test)       # Predicting test data with the model

acc_test = metrics.r2_score(y_test,pred_test)

print("R square score for test dataset for Ada Boost Regressor: ", acc_test)

ada_score = cross_val_score(ada,X_scaler,y,cv=5)

ada_m = ada_score.mean()

print("Cross val score for Ada Boost Regresor:",ada_m*100)


mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for ada boost Regressor:",rmse)
```

```
=====ADA BOOST REGRESSOR=====
R square score for training dataset for Ada Boost regressor:  0.5548688552911463
R square score for test dataset for Ada Boost Regressor:  0.47195775006876794
Cross val score for Ada Boost Regresor: 60.37085732954173
Root mean squared Error for ada boost Regressor: 321300.45832848374
```

5. XGradient Boosting Regressor

XGBRegressor

In [10]:

```
from xgboost import XGBRegressor

XGB=XGBRegressor()

XGB.fit(X_train,y_train)

pred_train = XGB.predict(X_train)       # Predicting training data with the model

acc_train = metrics.r2_score(y_train,pred_train)    # Training accuracy

print("R square score for training dataset for XGB: ", acc_train)

pred_test = XGB.predict(X_test)       # Predicting test data with the model

acc_test = metrics.r2_score(y_test,pred_test)

print("R square score for test dataset for XGB: ", acc_test)

xgb_score = cross_val_score(XGB,X_scaler,y,cv=5)

xgb_m = xgb_score.mean()

print("Cross val score for XGB:",xgb_m*100)


mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for XGB:",rmse)
```

```
[07:53:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
R square score for training dataset for XGB:  0.9003149123988741
R square score for test dataset for XGB:  0.8916273949013256
[07:53:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:53:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:53:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:53:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:53:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Cross val score for XGB: 81.59509576464453
```

6. Extra Trees Regressor

Model 6 - Extreme Gradient Boosting Regressor

```
xgb = XGBRegressor()

xgb.fit(X_train,y_train)

xgb.score(X_train,y_train)

pred_train = xgb.predict(X_train)

pred_test = xgb.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("=====XGBoosting REGRESSOR=====")

print("Training accuracy for XGradient Boost model:",score_train*100)

print("Testing accuracy for XGradient Boost model:",score_test*100)

cv_score = cross_val_score(xgb,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean*100} ")

# MSE - ignoring some outliers means larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for XGradient Boosting model:",rmse)
```

```
[10:05:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
=====XGBoosting REGRESSOR=====
Training accuracy for XGradient Boost model: 92.20843757213169
Testing accuracy for XGradient Boost model: 91.18479392429315
[10:05:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:05:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

<u>Hyper Parameter Tuning for XGradient Boost Regressor</u>

Tuning parameters for XGB

```
params_xgb = {'max_depth':[3,4,5,6],
              'learning_rate':(0.01,0.001,0.1),
              'min_child_weight':(0,1,2),
              'max_delta_step':(0,1,2)
              }

grd_xgb = GridSearchCV(XGB,params_xgb)

grd_xgb.fit(X_train,y_train)

print(grd_xgb.best_params_)
```

```
[07:27:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:27:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
{'learning_rate': 0.1, 'max_delta_step': 0, 'max_depth': 6, 'min_child_weight': 0}
```

```
XGB = grd_xgb.best_estimator_

XGB.fit(X_train,y_train)

XGB.score(X_train,y_train)

pred_train = XGB.predict(X_train)

pred_test = XGB.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("=====XGradient Boosting REGRESSOR=====")

print("Training accuracy for XGradient Boost model:",score_train*100)

print("Testing accuracy for XGradient Boost model:",score_test*100)

cv_score = cross_val_score(XGB,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean*100} ")

# MSE - ignoring some outliers means Larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for XGradient Boosting model:",rmse)
```

```
[07:29:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
=====XGradient Boosting REGRESSOR=====
Training accuracy for XGradient Boost model: 96.72905877107014
Testing accuracy for XGradient Boost model: 92.90096570221175
[07:29:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:29:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:29:17] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[07:29:17] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

- Key Metrics for success in solving a problem under consideration

Following metrics used for evaluation:

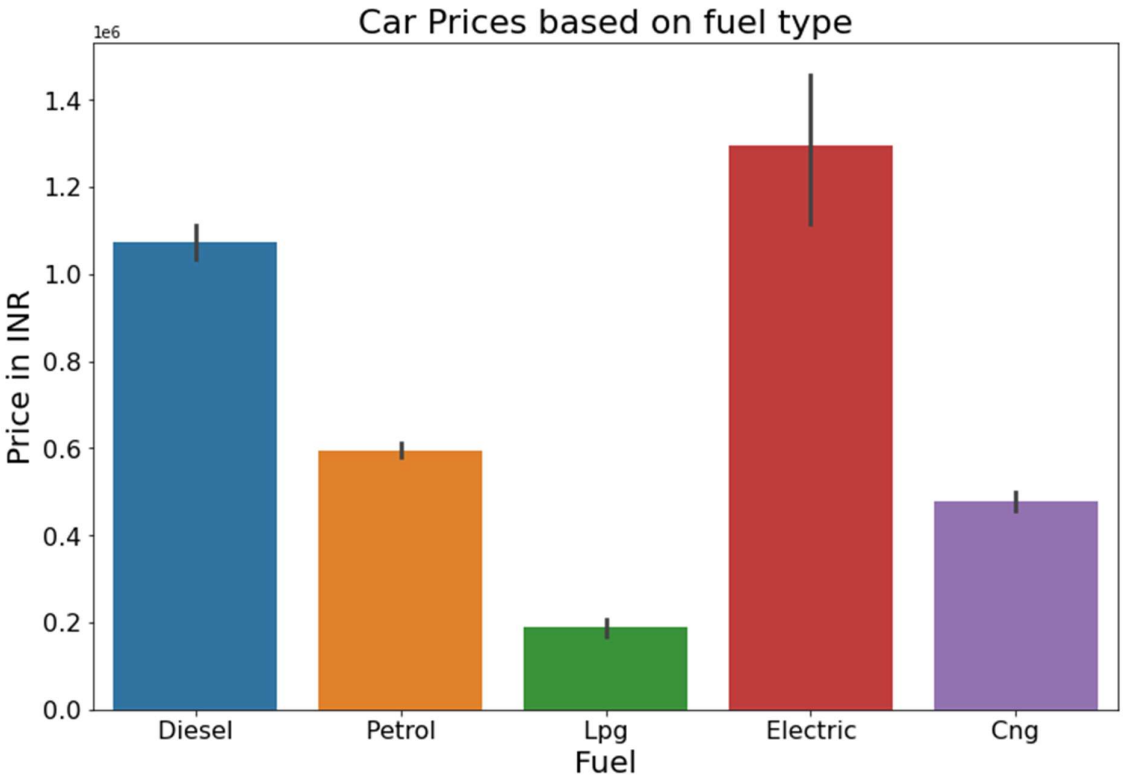
1. Root mean square error is one of the most commonly used measures

for evaluating the quality of predictions.

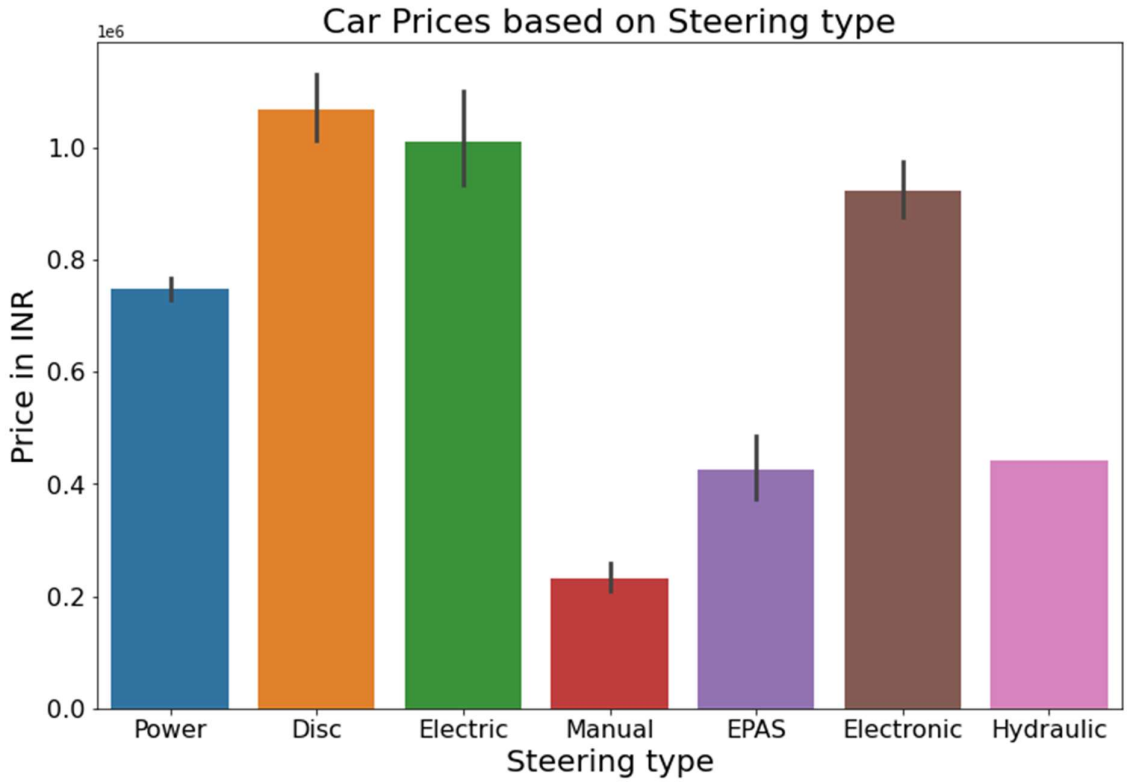
2. R2 score which tells us how accurately our model predicts a result, is going to be an important evaluation criterion

3. Cross-Validation Score

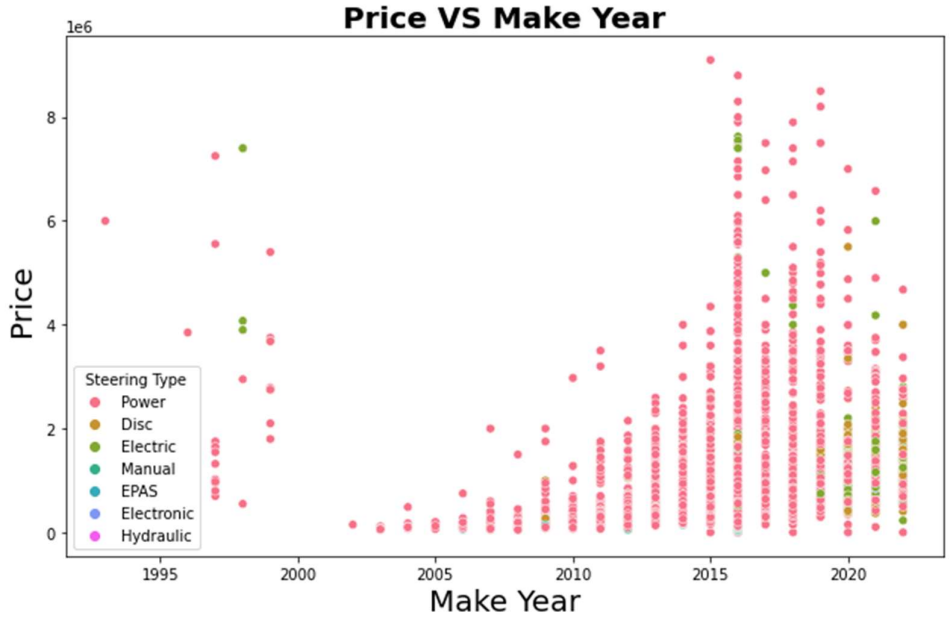
- Visualizations



- ✓ Only 29 electric vehicles out of 8704 cars and still electric vehicle are costliest. There are several factors that contribute to the higher cost of electric vehicles (EVs) compared to traditional gasoline-powered cars.
  - Battery technology: The battery is one of the most expensive components of an EV, and as the technology continues to improve, the cost may decrease
  - Low production volumes: Currently, there are relatively low production volumes for EVs compared to traditional vehicles, which can lead to higher costs due to economies of scale
- ✓ Diesel engines tend to retain their value better than petrol engines, which can contribute to higher prices for used diesel vehicles



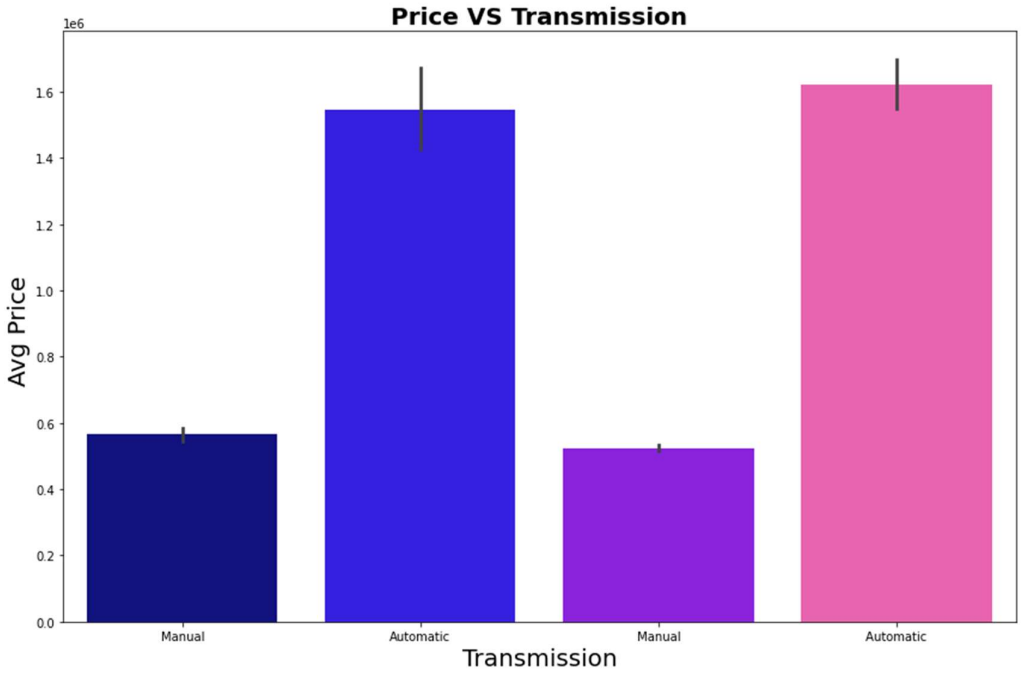
- ✓ The type of steering in a used car can have some impact on its value, but it is not typically a significant factor. The price of a used car is usually influenced more by factors such as age, make and model, mileage, condition, and history



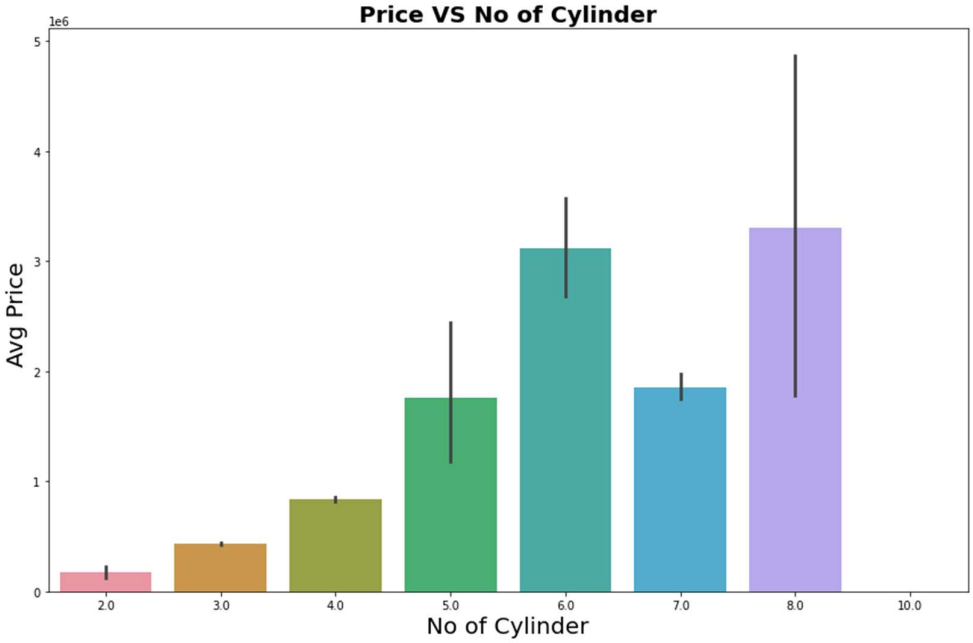
- ✓ Generally, older cars are cheaper, but their condition also affects their value

- ✓ A used car in excellent condition will be worth more than one in poor condition, even if it is newer

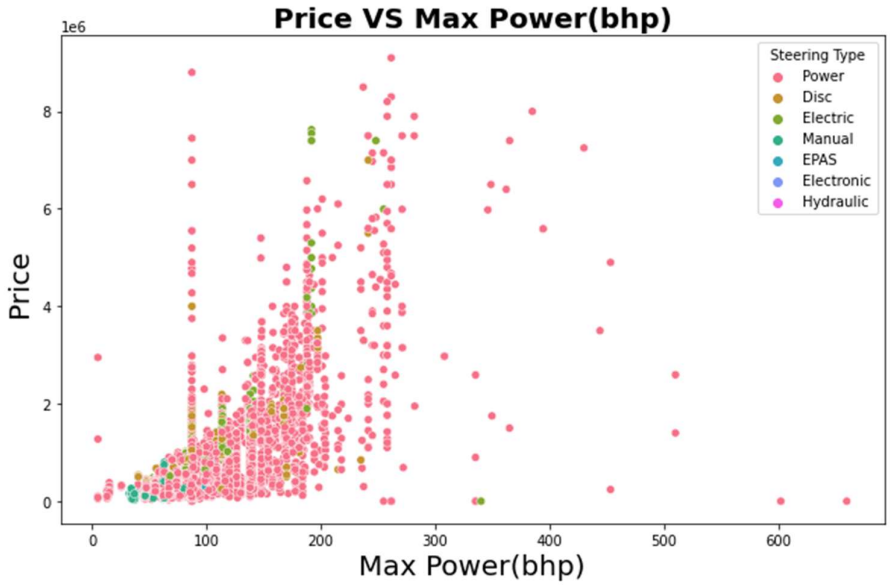
- ✓ As it can be seen that some power steering 2015 cars are costlier than 2020 manufactured cars



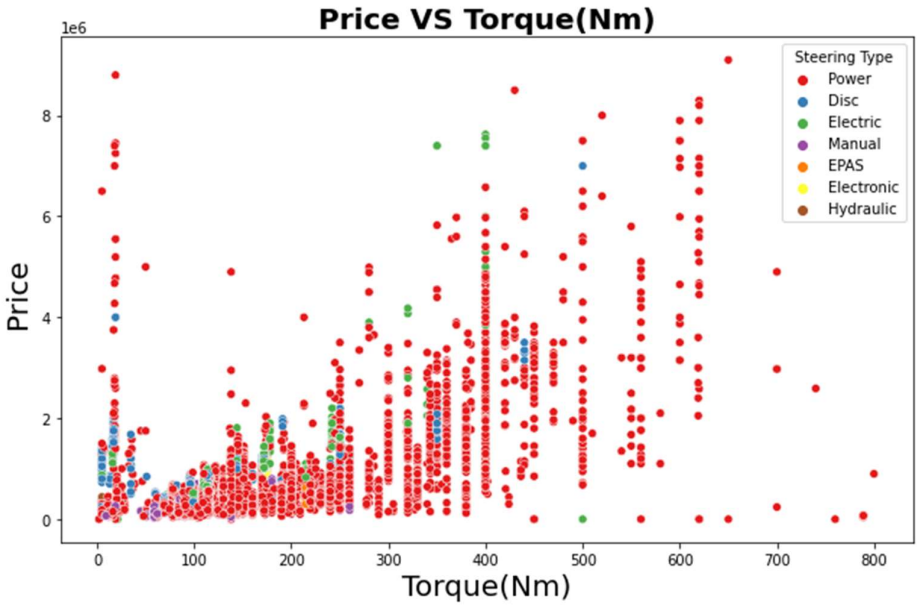
- ✓ Used cars with a manual transmission are less expensive than car with an automatic transmission as automatic transmissions tend to be more complex and expensive to repair



✓ In general, cars with more cylinders tend to be more powerful and perform better. Therefore, they can command a higher price in the used car market.



✓ Generally, cars with higher bhp are considered to be more powerful and sportier, and as a result, they are often priced higher in the used car market.



✓ Torque, which is a measure of a vehicle's rotational force, can affect a vehicle's acceleration and overall performance

✓ Vehicles with higher torque ratings typically have more powerful engines

✓ Hence it can be seen that high torque rating cars have higher price

4. CONCLUSION

- **Key Findings and Conclusions of the Study**

ML Algorithm	R2 Score		CV Score
	Train	Test	
Linear Regression	65.54%	66.84%	-1.06
K Neighbors Regressor	87.36%	83.48%	72.19%
Random Forest Regressor	98.54%	92.72%	85.75%
Ada Boost Regressor	55.48%	47.19%	60.37%
Extra Trees Regressor	99.98%	92.83%	81.66%
Extreme Gradient Boost Regressor(XGB)	90.03%	89.16%	81.59%
Hyper Parameter Tuning for XGB	96.72%	92.90%	85.28%

- ✓ The extreme Gradient Boosting (XGB) model gives the best result for the given dataset as there is least difference between training and test accuracy
- ✓ As we get the highest test accuracy at 89.16% and training accuracy at 90%

- **Learning Outcomes of the Study in respect of Data Science**

- ✓ XGB works well compared to other machine-learning algorithms
- ✓ XGBoost consists of the objective function and base learners
- ✓ The loss function is present in the objective function
- ✓ The loss function shows the difference between actual values and predicted values
- ✓ Regularisation term is used for showing how far is actual value away from the predicted value
- ✓ XGBoost considers many models which are known as base learners for predicting a single value

- ✓ Not all base learners are expected to have bad predictions so after summing up all of them bad predictions cancelled out by good prediction

- **<u>Limitations of this work and Scope for Future Work</u>**

  Deep learning algorithms may enhance the prediction of the price of the used cars and decrease the test error rate percentage.