August_Data Science and Business Analytics

Major Project
Exploratory Data Analysis

Insights/Suggestions

Guided By – Dhawani Shah Ma'a

In this major project, we were provided with the problem statement and the dataset on pre-owned car market.

"There is a huge demand of used cars in the Indian Market today. As sale of new car have slowed down in the recent past, the pre-owned car market has continued to grow over the past year and is larger than the new car market now. Consider this: In 2018-19, while new car sales were recorded at 3.6 million units, around4 million second-hand cars were bought and sold. There is a slowdown in new car sales and that could mean that the demand is shifting towards the pre-owned market. In fact, some car sellers replace their old cars with pre-owned cars instead of buying new ones."

The goal of the case is as follows:

- Perform EDA
- Build various Models to Predict the price (Build at least 2 models and compare the results and suggest which model works better)
- Insights/Suggestions

Dataset :

1. Name : Name of the car which includes Brand name and Model name
2. Location : The location in which the car is being sold or is available for purchase Cities.
3. Year : Manufacturing year of the car
4. Kilometers_driven : The total kilometers driven in the car by the previous owner(s) in KM.
5. Fuel_Type : The type of fuel used by the car. (Petrol, Diesel, Electric, CNG, LPG)
6. Transmission : The type of transmission used by the car.(Automatic / Manual)
7. Owner_Type : Type of ownership
8. Mileage : The standard mileage offered by the car company in kmpl or km/kg
9. Engine : The displacement volume of the engine in CC.
10. Power : The maximum power of the engine in bhp.
11. Colour : The colour of the car.

12. Seats : The number of seats in the car.
13. No. of Doors : The number of doors the car have.
14. New_Price : The price of a new car of the same model in INR Lakhs.(1 Lakh = 100, 000)
15. Price : The price of the used car in INR Lakhs (1 Lakh = 100, 000)

At First, we imported important libraries which were necessary in our project.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import math
import scipy.stats as stats
from scipy.stats import skew
plt.style.use('ggplot')

#Sklearn package's randomized data splitting function
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import r2_score

#To supress numerical display in scientific notations
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.max_rows', 300)
pd.set_option('display.max_colwidth',400)
pd.set_option('display.float_format', lambda x: '%.5f' % x)

#To supress Warnings
import warnings
warnings.filterwarnings('ignore')
```

Then we read and understood our dataset.

- Imported our dataset.
- Displayed the shape of our dataset.
- Overall information of our dataset.
- And the statistical description of our dataset.

```
cars = pd.read_csv('C:\\Users\\Hitesh Kumar\\Documents\\Projects\\Cars Price Prediction\\Cars.csv')
cars.sample(5)
```

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Colour	Seats	No. of Doors	Ne
525	Volkswagen Polo	Hyderabad	2013.00000	76476.00000	Petrol	Manual	First	16.47 kmpl	1198 CC	73.9 bhp	White	5.00000	4.00000	
351	Maruti Alto	Coimbatore	2010.00000	48105.00000	Petrol	Manual	First	19.7 kmpl	796 CC	46.3 bhp	White	5.00000	4.00000	
174	Maruti Swift	Kochi	2016.00000	45725.00000	Diesel	Manual	First	25.2 kmpl	1248 CC	74 bhp	White	5.00000	4.00000	
4775	Mini Clubman	Pune	2017.00000	8350.00000	Petrol	Manual	First	13.8 kmpl	1998 CC	192 bhp	White	5.00000	4.00000	44.
3760	Maruti Ritz	Hyderabad	2010.00000	79324.00000	Diesel	Manual	First	21.1 kmpl	1248 CC	73.9 bhp	Black/Silver	5.00000	4.00000	

```
cars.shape
```

```
(5961, 15)
```

```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5961 entries, 0 to 5960
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               5961 non-null   object
 1   Location           5950 non-null   object
 2   Year               5959 non-null   float64
 3   Kilometers_Driven  5953 non-null   float64
 4   Fuel_Type          5961 non-null   object
 5   Transmission       5934 non-null   object
 6   Owner_Type         5946 non-null   object
 7   Mileage            5959 non-null   object
 8   Engine             5944 non-null   object
 9   Power              5929 non-null   object
 10  Colour             5950 non-null   object
 11  Seats              5956 non-null   float64
 12  No. of Doors       5960 non-null   float64
 13  New_Price          824 non-null    object
 14  Price              5961 non-null   float64
dtypes: float64(5), object(10)
memory usage: 698.7+ KB
```

```
cars.describe()
```

	Year	Kilometers_Driven	Seats	No. of Doors	Price
count	5959.00000	5953.00000	5956.00000	5960.00000	5961.00000
mean	2013.38916	58711.10012	5.26914	4.11493	9.52810
std	3.24305	91712.20717	0.78905	0.34476	11.21438
min	1998.00000	171.00000	2.00000	2.00000	0.44000
25%	2011.50000	33931.00000	5.00000	4.00000	3.50000
50%	2014.00000	53000.00000	5.00000	4.00000	5.66000
75%	2016.00000	73000.00000	5.00000	4.00000	10.00000
max	2019.00000	6500000.00000	10.00000	5.00000	160.00000

Exploratory Data Analysis (EDA)

In this report, we performed exploratory data analysis (EDA) on the dataset to gain insights and understand the factors that influence car prices. The purpose of this analysis is to prepare the data for car price prediction modeling.

The dataset used for this analysis contains information about various cars, including their features and corresponding prices. It consists of the following variables:

Brand: The brand of the car (added after Feature Engineering).

Model: The model's name of the car (added after Feature Engineering).

Year: The manufacturing year of the car.

Mileage: The total distance traveled by the car in kilometers.

Fuel Type: The type of fuel used by the car (e.g., petrol, diesel).

Transmission: The type of transmission system in the car (e.g., manual, automatic).

Owner_Type: The number of previous owners of the car.

Price: The price of the car in the local currency.

Data Cleaning

Before proceeding with the analysis, we performed data cleaning to handle missing values, outliers, and inconsistencies in the dataset. The cleaning process involved:

1. Handling Missing Values: We checked for missing values in each variable and decided to either impute or remove them based on the proportion of missing values and their impact on the analysis.
2. Dealing with Outliers: We identified outliers using statistical methods such as box plots and removed them to prevent skewing the analysis results.
3. Handling Inconsistent Data: We checked for inconsistencies in variables such as fuel_type and transmission and resolved them by standardizing the values.

Analysis:

After cleaning the data, we performed various exploratory analyses to understand the relationships between different variables and their impact on car prices. Some key findings from the analysis are:

Year vs. Price: There is a positive correlation between the manufacturing year of a car and its price, indicating that newer cars tend to have higher prices.

Mileage vs. Price: There is a negative correlation between the mileage of a car and its price, suggesting that cars with lower mileage are generally priced higher.

Brand vs. Price: Different car brands exhibit varying price ranges, with luxury brands generally commanding higher prices compared to economy brands.

Fuel_Type and Transmission vs. Price: Cars with automatic transmission and diesel fuel type tend to have higher prices compared to their counterparts.

Data Pre-processing

In Data Pre-processing, we converted Engine, Power and Mileage columns from object data type to numerical data type by stripping strings like "Kmpl, km,/kg, cc, bhp".

```
cars['Mileage'] = cars['Mileage'].str.rstrip(" kmpl")
cars['Mileage'] = cars['Mileage'].str.rstrip(" km/kg")
```

```
cars['Engine'] = cars['Engine'].str.rstrip(" CC")
```

```
cars['Power'] = cars['Power'].str.rstrip(" bhp")
cars['Power'] = cars['Power'].replace(regex='null', value = np.nan)
```

```
#verify data
num=['Engine','Power','Mileage']
cars[num].sample(20)
```

	Engine	Power	Mileage
711	1968	167.7	18.33
3820	1199	73.9	22.07
4428	2494	102	12.8
4763	1591	121.3	13.0

We had checked for null values and replaced with nan.

checking for values containing 0.0 and replacing them by NAN

```
cars.query("Power == '0.0'")['Power'].count()
```

```
0
```

```
cars.query("Mileage == '0.0'")['Mileage'].count()
```

```
56
```

```
cars.loc[cars['Mileage']=='0.0', 'Mileage']=np.nan
```

```
cars.loc[cars['Engine']=='0.0', 'Engine'].count()
```

```
0
```

And filled seat column null values with the help of groupby and lambda function.

```
cars['Seats']=cars.groupby(['Name'])['Seats'].apply(lambda x:x.fillna(x.median()))
```

```
cars['Seats'].isnull().sum()
```

```
0
```

Processing New Price

We know that New_Price is the price of a new car of the same model in INR Lakhs.(1 Lakh = 100, 000)

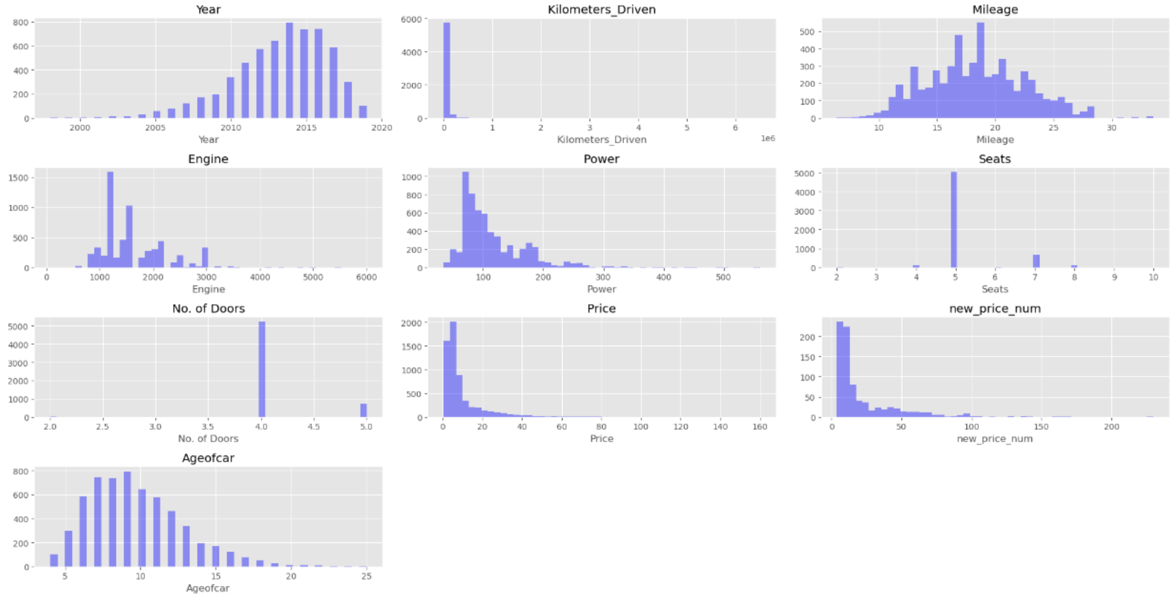
This column clearly has a lot of missing values. We will impute the missing values later. For now we will only extract the numeric values from this column.

```
import re

new_price_num = []

# Regex for numeric + " " + "Lakh"  format
regex_power = "^\d+(\.\d+)? Lakh$"

for observation in cars["New_Price"]:
    if isinstance(observation, str):
        if re.match(regex_power, observation):
            new_price_num.append(float(observation.split(" ")[0]))
        else:
            # To detect if there are any observations in the column
            # that we see in the sample output
            print(
                "The data needs furthur processing.mismatch ",
                observation,
            )
    else:
        # If there are any missing values in the New_Price column, 
        new_price_num.append(np.nan)
```

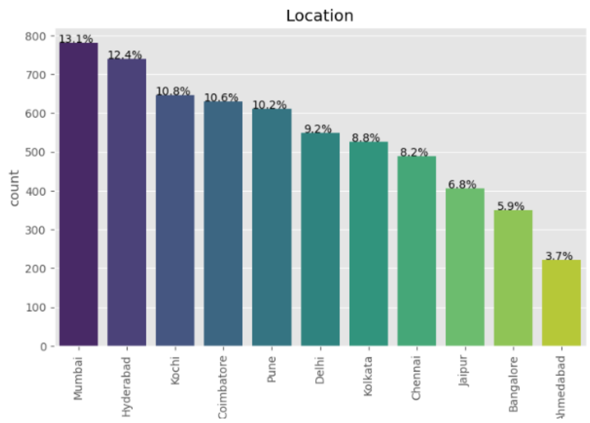
```
new_price_num = []

for observation in cars["New_Price"]:
    if isinstance(observation, str):
        if re.match(regex_power, observation):
            new_price_num.append(float(observation.split(" ")[0]))
        else:
            # Converting values in Crore to lakhs
            new_price_num.append(float(observation.split(" ")[0]) * 100)
    else:
        # If there are any missing values in the New_Price column, we add missing values to the new column
        new_price_num.append(np.nan)

# Add the new column to the data
cars["new_price_num"] = new_price_num

# Checking the new dataframe
cars.head(5)
```

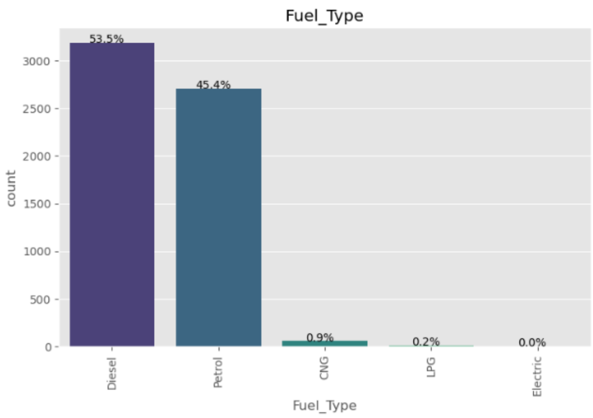
We have plotted the “ggplot” for all the columns.

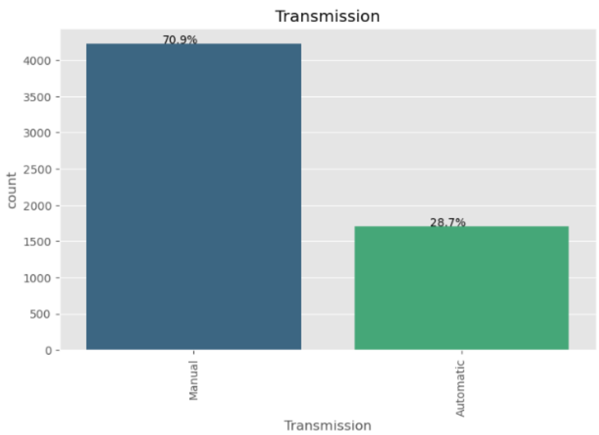
Years is left skewed. Years ranges from 1998- 2019 . Ageofcars 2 year old to 25 years old
Kilometer driven , median is ~53k Km and mean is ~58K. Max values seems to be 6500000. This is very high , and seems to be outlier. Need to analyze further.
Mileage is almost Normally distrubuited
Engine is right skewed and has outliers on higher and lower end
Power and Price are also right skewed.
Price 160 Lakh is too much for a used car. Seems to be an outlier.

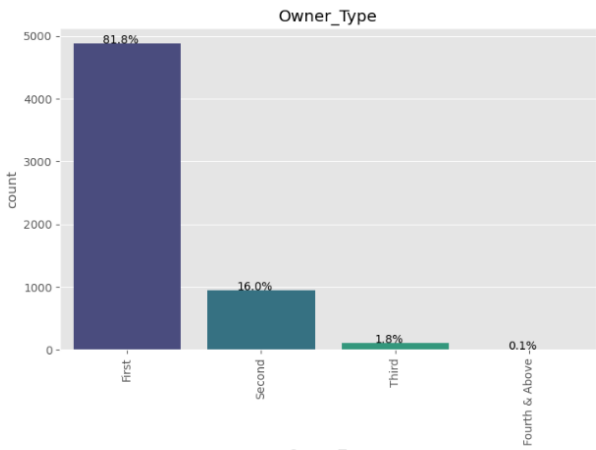


Year is left skewed and has outilers on lower side, This column can be dropped.
Kilometer_driven is right skewed.
Mileage is almost Normally distributed. Has few outliers on upper and lower side, need to check further.
Engine ,power and price are right skewed and has outliers on upper side.
Age of car is right skewed.









Car Profile

~71 % cars available for sell have manual Transmission.

~82 % cars are First owned cars.

~39% of car available for sale are from Maruti & Hyundai brands.

~53% of car being sold/avialable for purchase have fuel type as Diesel .

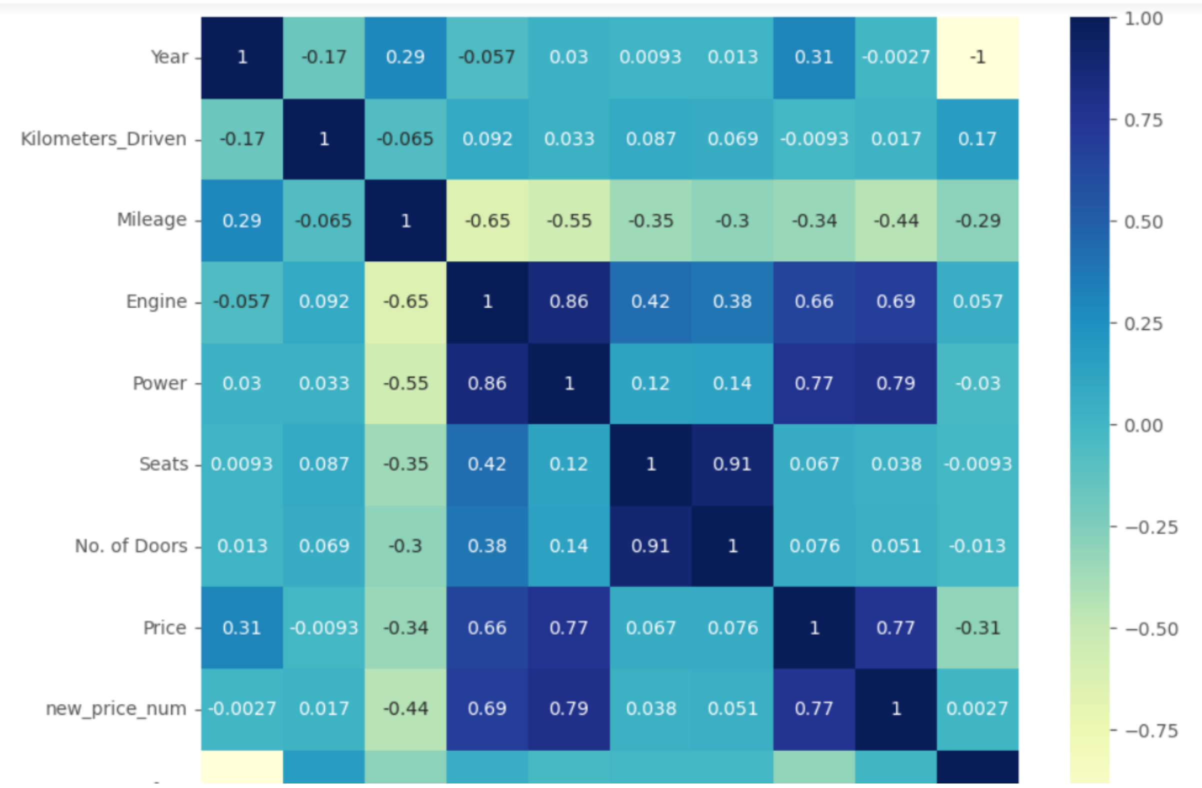
Mumbai has highest numbers of car availabe for purchase whereas Ahmedabad has least

Most of the cars are 5 seaters.

Car being sold/available for purchase are in 4 - 25 years old

~71% car are lower price range car.

After all the basic informations and pre processings we go for Bivariate & Multivariate Analysis



The exploratory data analysis provided valuable insights into the factors influencing car prices. The variables such as year, mileage, brand, fuel type, and transmission were found to have significant relationships with car prices. These findings will guide us in building an accurate car price prediction model based on the dataset.

Further analysis and modeling can be performed to develop a predictive model that accurately estimates car prices based on these influential factors.

Feature Engineering

We performed Feature Engineering in pre-owned car market Price Prediction model. Feature engineering is a machine learning technique that leverages data to create new variables that aren't in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of **simplifying and speeding up data transformations** while also **enhancing model accuracy**. Feature engineering is required when working with machine learning models. Regardless of the data or architecture, a terrible feature will have a direct impact on your model.

First we converted the data types into category and float.

Converting Datatype

```
cars["Fuel_Type"] = cars["Fuel_Type"].astype("category")
cars["Transmission"] = cars["Transmission"].astype("category")
cars["Owner_Type"] = cars["Owner_Type"].astype("category")
cars["Mileage"] = cars["Mileage"].astype('float')
cars["Power"] = cars["Power"].astype('float')
cars["Engine"]=cars["Engine"].astype('float')
```

We can observe here that "Fuel_Type, Transmission, Owner_type" has been changed to category type and "Mileage, Power, Engine" has been converted to float type.

We calculated age of cars and added it as a new feature named "Ageofcar".

```
#Processing Years to derive age of car

cars['Current_year']=2023
cars['Ageofcar']=cars['Current_year']-cars['Year']
cars.drop('Current_year',axis=1,inplace=True)
cars.head()
```

ers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Colour	Seats	No. of Doors	New_Price	Price	new_price_num	Ageofcar
000.00000	Diesel	Manual	Third	12.05000	2179.00000	120.00000	Black/Silver	8.00000	5.00000	NaN	6.00000	NaN	11.00000
678.00000	Petrol	Manual	First	21.10000	998.00000	100.00000	Others	5.00000	4.00000	NaN	8.32000	NaN	5.00000
000.00000	Diesel	Manual	First	11.68000	2498.00000	112.00000	White	7.00000	5.00000	NaN	4.00000	NaN	10.00000

We have extracted the brand and model names from Names column.

```
#Extracting brand name from Names column
cars[['Brand', 'Model']] = cars['Name'].str.split(n=1, expand=True)
```

```
cars.Brand.unique()
```

```
array(['Mahindra', 'Maruti', 'Hyundai', 'Toyota', 'Honda', 'Chevrolet',
       'Audi', 'Skoda', 'Renault', 'Land', 'BMW', 'ISUZU', 'Jaguar',
       'Mercedes-Benz', 'Volkswagen', 'Tata', 'Mitsubishi', 'Ford',
       'Nissan', 'Volvo', 'Fiat', 'Porsche', 'Mini', 'Datsun', 'Jeep',
       'Force', 'Isuzu', 'Smart', 'Lamborghini', 'Bentley'], dtype=object)
```

We have observed that some brand names are incorrect and need some changes, so we have changed and corrected the brand names of the car.

```
#changing brand names
cars.loc[cars.Brand == 'ISUZU','Brand']='Isuzu'
cars.loc[cars.Brand=='Mini','Brand']='Mini Cooper'
cars.loc[cars.Brand=='Land','Brand']='Land Rover'
```

Then we analysed the dataset to know which car brand was most available for purchased/sold for customers.

```
cars.groupby(cars.Brand).size().sort_values(ascending =False)
```

```
Brand
Maruti           1189
Hyundai          1100
Honda             601
Toyota            410
Mercedes-Benz     318
Volkswagen        315
Ford              298
Mahindra          272
BMW               267
Audi              236
Tata              184
Skoda             173
Renault           143
Chevrolet         113
Nissan             91
Land Rover         57
Jaguar             40
Mitsubishi         27
Mini Cooper        26
Fiat               25
Volvo              21
Porsche            18
Jeep               15
Datsun             13
Isuzu               3
Force               3
Lamborghini         1
Smart               1
Bentley             1
```

As the outcome displayed, there are 29 unique brands in the dataset. Maruti brand is most available for purchase/sold, followed by Hyundai.

Then for the proper prediction of the model we have handled the missing values in the dataset.

Handling Missing Values

```
cars.isnull().sum()
```

```
Name                    0
Location               11
Year                    2
Kilometers_Driven       8
Fuel_Type               0
Transmission           27
Owner_Type             15
Mileage                58
Engine                 17
Power                 135
Colour                 11
Seats                   0
No. of Doors            1
New_Price            5137
Price                   0
new_price_num        5137
Ageofcar                2
Brand                   0
Model                   0
dtype: int64
```

There were many missing values in the dataset, if we dropped all those values then our dataset will be of no use as it will only contain less than 1000 rows which can lead to inaccuracy and biased results. So, we imputed them by median with the help of grouping and lambda function.

We can start filling missing values by grouping name and year and fill in missing values. with median.

```
cars.groupby(['Name','Year'])['Engine'].median().head(30)
```

```
cars['Engine']=cars.groupby(['Name','Year'])['Engine'].apply(lambda x:x.fillna(x.median()))
cars['Power']=cars.groupby(['Name','Year'])['Power'].apply(lambda x:x.fillna(x.median()))
cars['Mileage']=cars.groupby(['Name','Year'])['Mileage'].apply(lambda x:x.fillna(x.median()))
```

```
cars.groupby(['Brand','Model'])['Engine'].median().head(10)
```

```
Brand  Model
Audi   A3       1968.00000
       A4       1968.00000
       A6       1968.00000
       A7       2967.00000
       A8       2967.00000
       Q3       1968.00000
       Q5       1968.00000
       Q7       2967.00000
       RS5      2894.00000
       TT       1984.00000
Name: Engine, dtype: float64
```

As we can see most of the model have same engine size and instead of just applying median , grouping with model and year that should give me more granularity, and near to accurate Engine values.

```
#chosing Median to fill the the missing value as there are many outliers,
#grouping by model and year to get  more granularity and more accurate Engine and then fillig with median
cars['Engine']=cars.groupby(['Brand','Model'])['Engine'].apply(lambda x:x.fillna(x.median()))
```

```
#chosing Median to fill the the missing value as there are many outliers,
#grouping by model to get more granularity and more accurate Engine
cars['Power']=cars.groupby(['Brand','Model'])['Power'].apply(lambda x:x.fillna(x.median()))
```

```
#chosing Median to fill the the missing value as there are many outliers,
#grouping by model to get more granularity and more accurate Engine
cars['Mileage']=cars.groupby(['Brand','Model'])['Mileage'].apply(lambda x:x.fillna(x.median()))
```

```
cars.isnull().sum()
```

```
Name                0
Location            0
Year                0
Kilometers_Driven   0
Fuel_Type           0
Transmission        0
Owner_Type          0
Mileage             0
Engine              0
Power               0
Colour              0
Seats               0
No. of Doors        0
Price               0
new_price_num       0
Ageofcar            0
Brand               0
Model               0
dtype: int64
```

```
cars.shape
```

```
(5790, 18)
```

Finally done with all missing values handling

Model Building

For model building first we checked, whether the distribution is skewed or not. As the distribution was skewed, it was necessary to use log transformation before model building.

```
#check distribution if skewed. If the distribution is skewed, it is necessary to use log transform before model building
cols_to_log = cars.select_dtypes(include=np.number).columns.tolist()
for colname in cols_to_log:
    sns.distplot(cars[colname], kde=True)
    plt.show()
```

```
#since the distributions are right skewed, using log transform will help in normalization

def Perform_log_transform(df,col_log):
    "#Perform Log Transformation of dataframe , and list of columns"
    for colname in col_log:
        df[colname + '_log'] = np.log(df[colname])
    #df.drop(col_log, axis=1, inplace=True)
    df.info()
```

```
Perform_log_transform(cars,['Kilometers_Driven','Price'])
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5790 entries, 0 to 5960
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Name                  5790 non-null   object
 1   Location              5790 non-null   category
 2   Year                  5790 non-null   float64
 3   Kilometers_Driven     5790 non-null   float64
 4   Fuel_Type             5790 non-null   category
 5   Transmission          5790 non-null   category
 6   Owner_Type            5790 non-null   category
 7   Mileage               5790 non-null   float64
 8   Engine                5790 non-null   float64
 9   Power                 5790 non-null   float64
 10  Colour                5790 non-null   object
 11  Seats                 5790 non-null   float64
 12  No. of Doors          5790 non-null   float64
 13  Price                 5790 non-null   float64
 14  new_price_num         5790 non-null   float64
 15  Ageofcar              5790 non-null   float64
 16  Brand                 5790 non-null   category
 17  Model                 5790 non-null   object
 18  Kilometers_Driven_log 5790 non-null   float64
 19  Price_log             5790 non-null   float64
dtypes: category(5), float64(12), object(3)
```

Before proceeding to Model Building, encoding the categorical and object variables is always a crucial step. For which we used get_dummies() .

```
X = cars.drop(['Price','Price_log'], axis=1)
Y = cars[['Price_log']]
```

Creating dummy variables

```
def encode_cat_vars(x):
    x = pd.get_dummies(
        x,
        columns=x.select_dtypes(include=["object", "category"]).columns.tolist(),
        drop_first=True,
    )
    return x
```

```
#Dummy variable creation is done before spliting the data , so all the different categories are covered
#create dummy variable
X = encode_cat_vars(X)
X.head()
```

	Kilometers_Driven	Mileage	Engine	Power	Seats	No. of Doors	Ageofcar	Kilometers_Driven_log	Location_Bangalore	Location_Chennai	...	Fuel_Type
0	99000.00000	12.05000	2179.00000	120.00000	8.00000	5.00000	11.00000	11.50288	0	0	...	
1	18678.00000	21.10000	998.00000	100.00000	5.00000	4.00000	5.00000	9.83510	0	0	...	
2	197000.00000	11.68000	2498.00000	112.00000	7.00000	5.00000	10.00000	12.19096	1	0	...	
3	45000.00000	24.00000	1120.00000	70.00000	5.00000	4.00000	9.00000	10.71442	0	0	...	
4	65000.00000	12.80000	2494.00000	102.00000	8.00000	5.00000	12.00000	11.08214	0	0	...	

5 rows × 28 columns

Then we split the data into train and test sets for model building.

```
#splitting the data in train and test
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3, random_state=0)
```

```
print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("y_train:",Y_train.shape)
print("y_test:",Y_test.shape)
```

```
X_train: (4053, 28)
X_test: (1737, 28)
y_train: (4053, 1)
y_test: (1737, 1)
```

For the first model prediction, we used the Linear Regression.

Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
linReg = LinearRegression()
```

```
#training the model

linReg.fit(X_train,Y_train)
```

And it gives the R-squared Score of 0.88085041918972785

```
#checking the accuracy

from sklearn.metrics import *
linReg_r2 = r2_score(Y_test,y_pred)
linReg_r2
```

```
0.8808504918972785
```

For the second model prediction, we used the Decision tree regressor.

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
```

```
dt = DecisionTreeRegressor()
```

```
dt.fit(X_train,Y_train)
```

And it gives the R-squared Score of 0.8673641667746893

```
dt_r2 = r2_score(Y_test,y_dt_pred)
dt_r2
```

```
0.8673641667746893
```

For the third model prediction, we used the Random forest Regressor.

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf = RandomForestRegressor()
```

```
rf.fit(X_train,Y_train)
```

And it gives the R-squared Score of 0.9260023704614854

```
rf_r2 = r2_score(Y_test,y_rf_pred)
rf_r2
```

```
0.9260023704614854
```

At last, we compared the all three models to get the best result, and that was **Random Forest Regressor** with the R-squared score of **0.9260023704614854.**

Comparing Models ¶

```
print("The R-squared Score for Linear Regression Model is :", linReg_r2)
print("The R-squared Score for Decision Tree Regressor Model is :", dt_r2)
print("The R-squared Score for Random Forest Regressor Model is :", rf_r2)
```

```
The R-squared Score for Linear Regression Model is : 0.8808504918972785
The R-squared Score for Decision Tree Regressor Model is : 0.8673641667746893
The R-squared Score for Random Forest Regressor Model is : 0.9260023704614854
```

On comparision the best fitted model was Random Forest Regressor with the R-squared score of ~93%

Insights and Suggestions

1. **Feature Importance**: The accuracy of Random Forest model suggests that features such as Year, Kilometers Driven, Mileage, and Engine have the most significant impact on car prices. Sellers should emphasize these aspects when listing their cars.

2. **Location Matters**: The city where the car is sold affects its price. Consider regional variations in pricing when selling or purchasing a used car.

3. **Data Collection**: Collect more data on vehicle conditions, service history, and additional features to improve prediction accuracy.

4. **Interactive Tools**: Develop user-friendly tools for potential buyers and sellers to estimate car prices easily, incorporating the predictive model.

5. **Regular Updates**: Continuously update the model with new data to adapt to changing market dynamics.

Conclusion

The used car sales prediction project is crucial in the growing Indian pre-owned car market. Our analysis and model comparison indicate that the Random Forest Regressor offers the best price prediction accuracy. The insights and suggestions provided can help both buyers and sellers make informed decisions in this evolving market.