CAR: PRICE PREDICTION

Submitted By

Amruta Shah

Problem Statement:

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.

This project contains two phases: -

a) Data Collection Phase
b) Model Building Phase

Business Goal:

a) Data Collection Phase:-

You have to scrape at least 5000 used cars data. You can scrape more data as well, it's up to you. more the data better the model In this section You need to scrape the data of used cars from websites (Olx, cardekho, Cars24 etc.) You need web scraping for this. You have to fetch data for different locations. The number of columns for data doesn't have limit, it's up to you and your creativity. Generally, these columns are Brand, model, variant, manufacturing year, driven kilometers, fuel, number of owners, location and at last target variable Price of the car. This data is to give you a hint about important variables in used car model. You can make changes to it, you can add or you can remove some columns, it completely depends on the website from which you are fetching the data. Try to include all types of cars in your data for example- SUV, Sedans, Coupe, minivan, Hatchback.

b) Model Building Phase:-

After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science. Include all the steps like.

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best mode

Technical Requirements:

• Data contains 5024 entries each having 7 variables.

• Data contains Null values. You need to treat them using the domain knowledge and your own understanding.
• Extensive EDA has to be performed to gain relationships of important variable and price.
• Data contains numerical as well as categorical variable. You need to handle them accordingly.
• You have to build Machine Learning models, apply regularization and determine the optimal values of Hyper Parameters. If any.
• Two task is their data scrapping by using web scarping and saved it as csv file & then use this file to build the model.

Methodology:

The steps followed in this work, right from the dataset preparation to obtaining results are represented in Fig.

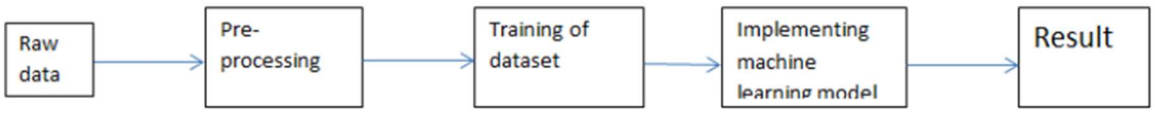


Fig1: Steps followed for obtaining results

Data Analysis:

In this project, we have a dataset which has the details of the prospective properties to buy used cars from different sources from the market.

The given dataset contains 5024rows × 7 columns. The column names like Price, Kilometre, owner, Type, Fuel etc.

EDA (Exploratory Data Analysis):

We should first perform an EDA as it will connect us with the dataset at an emotional level and yes, of course, will help in building good hypothesis function. EDA is a very crucial step. It gives us a glimpse of what our data set is all about, its uniqueness, its anomalies and finally it summarizes the main characteristics of the dataset for us. In order to perform EDA, we will require the following python packages.

Import libraries:

Let's use collected data set to solve the problem. For that we need to import some necessary python libraries.

```
import pandas as pd        # for data manipulation
import numpy as np         # for mathematical calculations
import seaborn as sns      # for data visualization

import matplotlib.pyplot as plt  #for graphical analysis
%matplotlib inline

from scipy.stats import zscore # to remove outliers

from sklearn.preprocessing import StandardScaler  # for normalize the model
from sklearn.preprocessing import LabelEncoder  # to convert object into int


from sklearn.model_selection import train_test_split  # for train and test model

import warnings                    # to ignore any warnings
warnings.filterwarnings("ignore")

from sklearn import metrics  # for model evaluation
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, accuracy_score, confusion_matrix, classificat
```

Once we have imported the packages successfully, we will move on to importing our dataset.

Load Dataset:

This collected data is in the .csv form so we need to use Pandas. read method to read the data.

```
data=pd.read_csv('used_cars.csv')  # read the data
data
```

	Name	Model	Fuel	Kilometer	Type	Onwer	Price
0	2013 Maruti Ritz	ZXI Manual	Petrol	14,047 km	Manual	1st	₹3,83,999
1	2016 Maruti Baleno	DELTA 1.2 K12 Manual	Petrol	11,510 km	Manual	2nd	₹5,06,399
2	2012 Honda Accord	2.4 MT Manual	Petrol	50,161 km	Manual	2nd	₹5,20,199
3	2019 Maruti Swift	ZXI AMT Automatic	Petrol	15,445 km	Automatic	2nd	₹6,73,399
4	2014 Maruti Wagon R 1.0	LXI Manual	Petrol	13,900 km	Manual	1st	₹3,38,799
...	...	...	...	...	...	...	...
5019	Hyundai I10 (2010)	Era	PETROL	42000.0 KM	MANUAL	2nd	₹ 2,25,000
5020	Hyundai Elite I20 (2015)	Sportz (O) 1.2	PETROL	111000.0 KM	MANUAL	2nd	₹ 5,99,000
5021	Bmw 3 Series (2012)	320d Luxury Line	DIESEL	85000.0 KM	AUTOMATIC	2nd	₹ 11,99,000
5022	Maruti Suzuki Swift Dzire (2015)	Vdi BSIV	DIESEL	71000.0 KM	MANUAL	1st	₹ 4,25,000
5023	Maruti Suzuki Ritz (2010)	VDI (ABS) BS-IV	DIESEL	68000.0 KM	MANUAL	2nd	₹ 3,25,000

5024 rows × 7 columns

Dimensions of Dataset:

The dataset has been successfully imported. Let's have a look at the dataset. head () gives us a glimpse of the dataset. It can be considered similar to select * from database table limit 5 in SQL. Let's go ahead and explore a little bit more about the different fields in the dataset. info () gives uses all the relevant information on the dataset. If your dataset has more numerical variables, consider using describe () too to summarize data along mean, median, standard variance, variance, unique values, frequency etc. isna (). sum () gives us sum of null values are present in the dataset. See below screenshot.

```
# check the no. of rows & column & finding the count of missing value
print(data.shape)
print("-"*60)
print(data.isna().sum())
#printing the summary of data type
data.info()
```

```
(5024, 7)
------------------------------------------------------------
Name           3
Model         10
Fuel           3
Kilometer      3
Type          35
Onwer          6
Price        456
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5024 entries, 0 to 5023
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Name       5021 non-null   object
 1   Model      5014 non-null   object
 2   Fuel       5021 non-null   object
 3   Kilometer  5021 non-null   object
 4   Type       4989 non-null   object
 5   Onwer      5018 non-null   object
 6   Price      4568 non-null   object
dtypes: object(7)
memory usage: 274.9+ KB
```

The number of null values present is as above.

Dataset has object types. Object type in pandas is similar to strings. Now let's try to classify these columns as Categorical, Ordinal or Numerical/Continuous.
In this our dataset has we have all **Ordinal Variables, Categorical & Numerical or Continuous Variables only.**

Categorical Variables:
Categorical variables are those data fields that can be divided into definite groups. In this case, Type is categorical variables are present.

Ordinal Variables:
Ordinal variables are the ones that can be divided into groups, but these groups have some kind of order. Like, high, medium, low. Dependents field can be considered ordinal since the data can be clearly divided into 4 categories: 0, 1, 2, 3 and there is a definite ordering also. In this case we have Fuel, Owner has ordinal variable.

Numerical or Continuous Variables:
Numerical variables are those that can take up any value within a given range. In this case Price.

Statistical Summary:

In the raw data, there can be various types of underlying patterns which also gives an in-depth knowledge about subject of interest and provides insights about the problem. But caution should be observed with respect to data as it may contain null values, or redundant values, or various types of ambiguity, which also demands for pre-processing of data. Dataset should therefore be explored as much as possible. Various factors important by statistical means like mean, standard deviation, median, count of values and maximum value etc. are shown in Fig below.

```
# Lets understand data at high level check the stastics of dataset
data.describe(include='all')
```

	Name	Model	Fuel	Kilometer	Type	Onwer	Price
count	3762	3762	3762	3.762000e+03	3762	3762.000000	3.762000e+03
unique	973	1448	7	NaN	2	NaN	NaN
top	0	vxi	petrol	NaN	manual	NaN	NaN
freq	477	102	2461	NaN	2437	NaN	NaN
mean	NaN	NaN	NaN	5.067751e+04	NaN	1.070973	8.423900e+05
std	NaN	NaN	NaN	5.995798e+04	NaN	0.275784	8.919886e+05
min	NaN	NaN	NaN	1.010000e+02	NaN	1.000000	3.000000e+04
25%	NaN	NaN	NaN	2.626075e+04	NaN	1.000000	4.626242e+05
50%	NaN	NaN	NaN	4.678800e+04	NaN	1.000000	5.724495e+05
75%	NaN	NaN	NaN	6.818700e+04	NaN	1.000000	9.339248e+05
max	NaN	NaN	NaN	2.950000e+06	NaN	4.000000	1.680000e+07

Observations: 1) Null values are present
2) We have categorical data type (object type)
3) Outliers & Skewed data are present in the dataset
4) In some columns there is too much difference between mean and std. deviation.

Data Visualization:

We now have a basic idea about the data. We need to extend that with some visualizations. We are going to look at three types of plots:

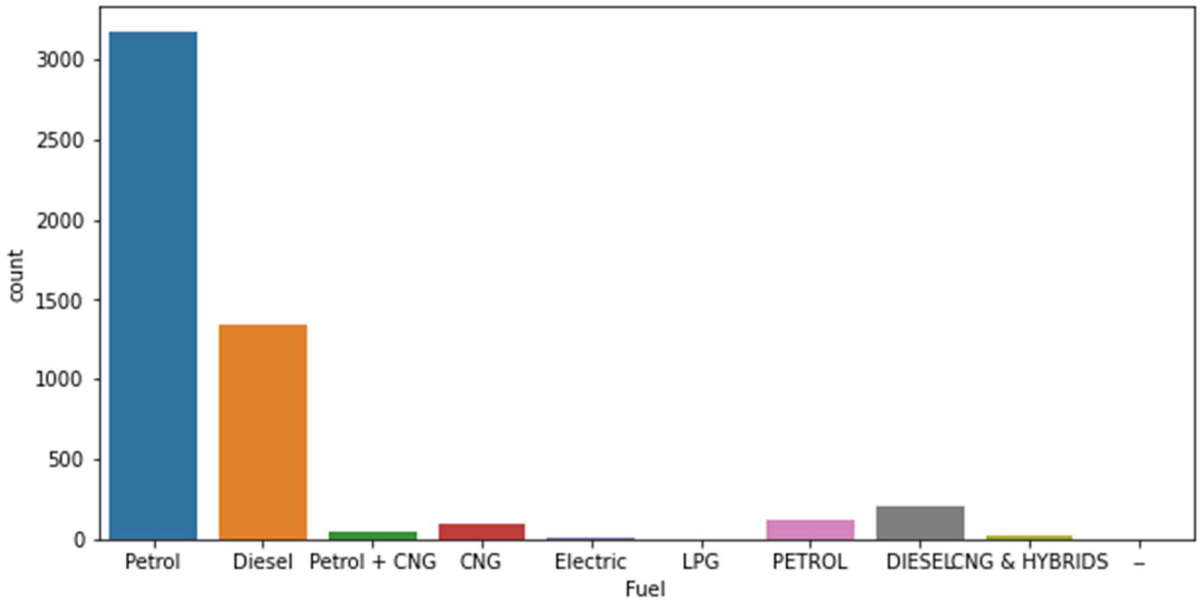
1.Univariate plots to better understand each variable.
2.Bivariate plots to find relationship between two variables,
3.Multivariate plots to better understand the relationships between variables.

Univariate Plots:

We start with some univariate plots, that is, plots of each individual variable. Given that the input variables are numeric, we can create box or count plots of each. Now we are all set to perform Univariate Analysis. Univariate analysis involves analysis of one variable at a time.

Let's say "Fuel" then we will analyse "Owner" & others field in the dataset. The analysis is usually summarized in the form of count. For visualization, we have many options such as frequency tables, bar graphs, pie charts, histograms etc. We will stick to count charts.

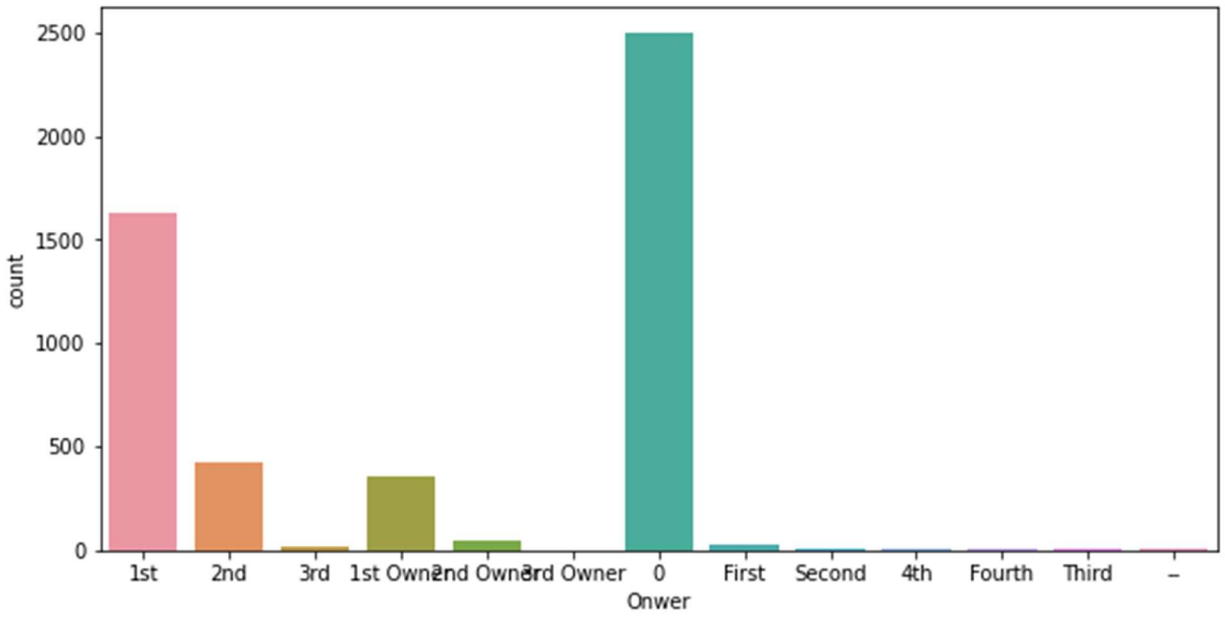
```
#plot each class frequency
plt.figure(figsize=(10,5))
sns.countplot(x='Fuel',data=data)
plt.show()
print(data['Fuel'].value_counts())
```




```
Petrol           3174
Diesel           1341
DIESEL            202
PETROL            121
CNG                98
Petrol + CNG       52
CNG & HYBRIDS      23
Electric            7
-
```

From Graph we can see there are maximum used cars fuel type are petrol & least used cars fuel for sold is lpg. Also we see there are some "--" type of fuel is present in data set so let's deal with it in pre-processing step.

```
#plot each class frequency
plt.figure(figsize=(10,5))
sns.countplot(x='Onwer',data=data)
plt.show()
print(data['Onwer'].value_counts())
```




```
0             2499
1st           1630
2nd            427
1st Owner      352
2nd Owner       44
First           25
3rd             18
Second           7
4th              5
```

As we see from graph that we have 0 null values & '-' unstructured data also so let's deal with it in pre-processing step.

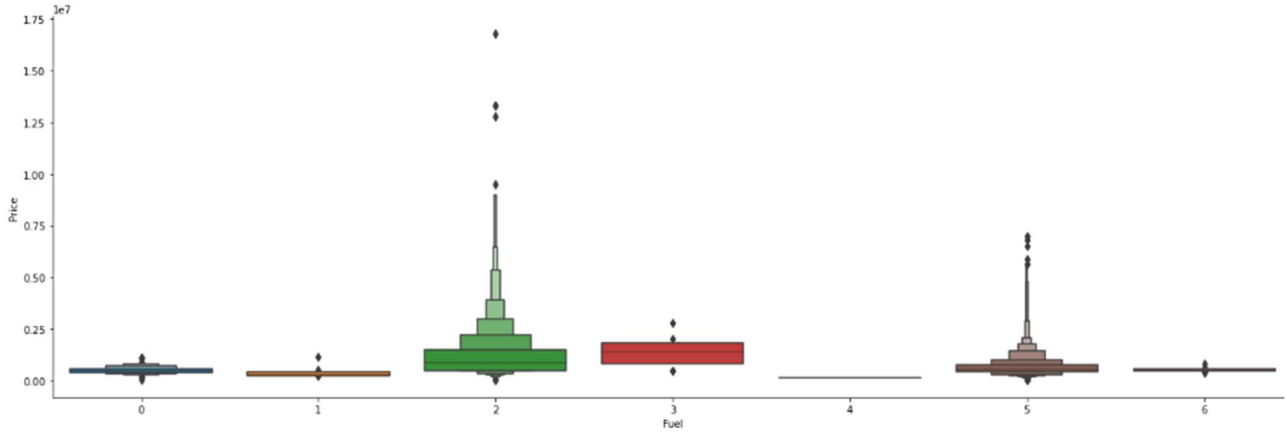
Similarly, I have plotted for the other independent variables and concluded in the notebook.

Bivariate Plot:

Now let's find some relationship between two variables, particularly between the target variable and a predictor variable from the dataset. Formally, this is known as bivariate analysis. Hear we have target variable is Price, so let's check this relationship between two variables and other independent variables. For visualization, we will be using seaborn. countplot (). It can be considered similar to the histogram for categorical variables

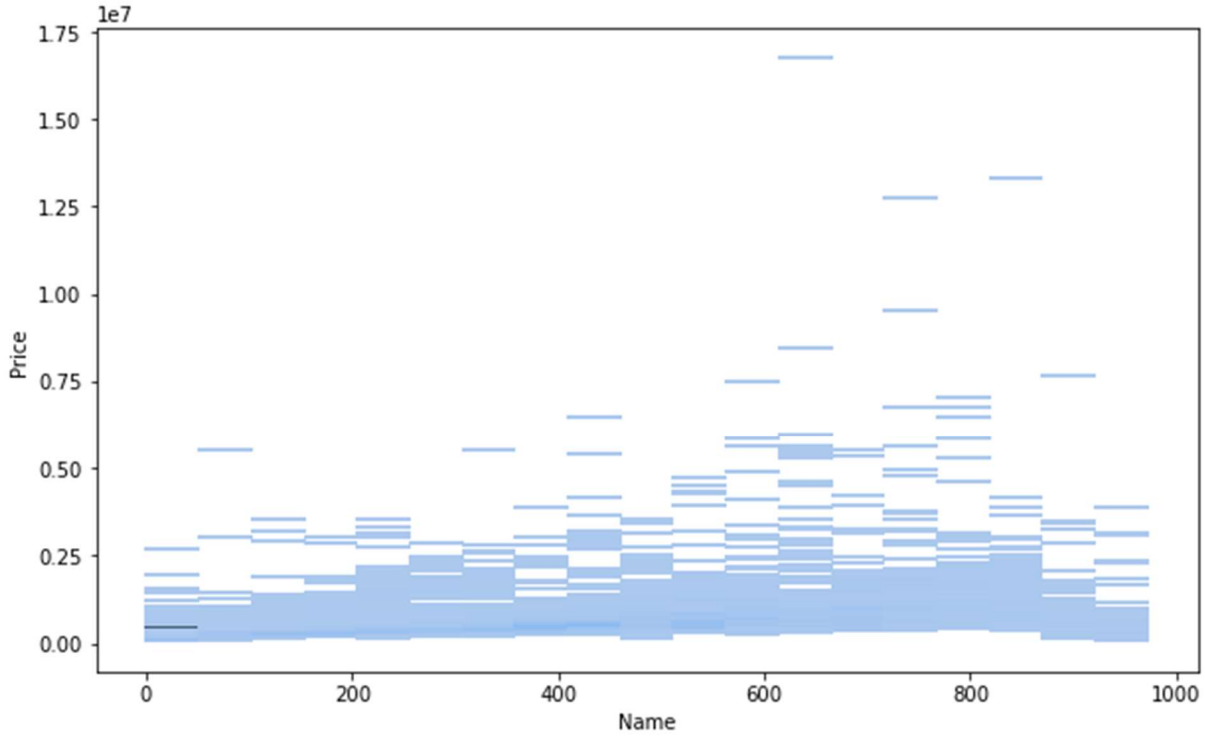
```
#Bivariant graph
plt.figure(figsize =(10, 6))
sns.catplot(x ='Fuel', y ='Price', data = data, kind = "boxen", height = 6, aspect = 3)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



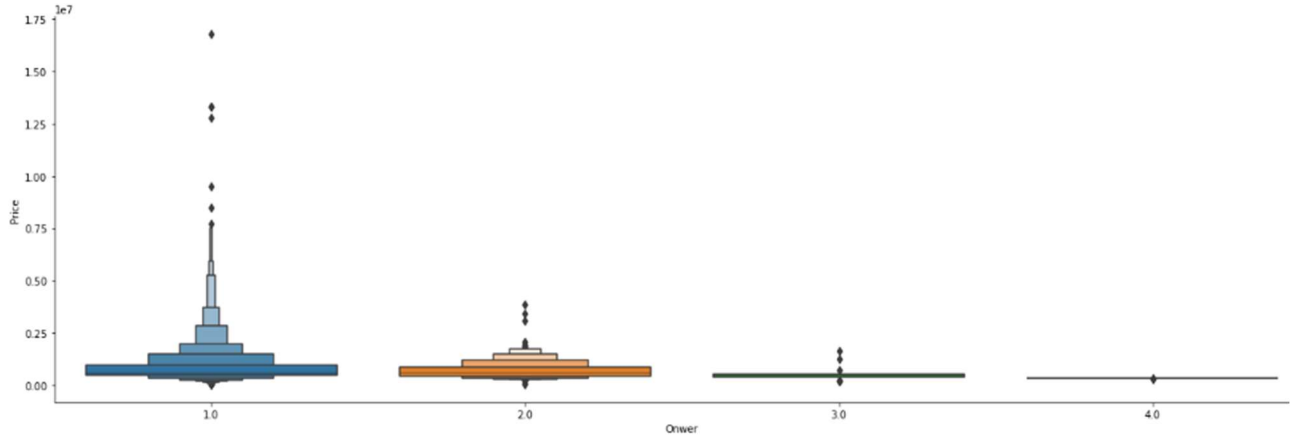
From graph we can see the type 2 has the maximum peice

```
#Bivariant graph
plt.figure(figsize =(10, 6))
sns.histplot(x ='Name', y ='Price', data = data)
plt.show()
```



```
#Bivariant graph
plt.figure(figsize =(10, 6))
sns.catplot(x ='Onwer', y ='Price', data = data, kind = "boxen", height = 6, aspect = 3)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



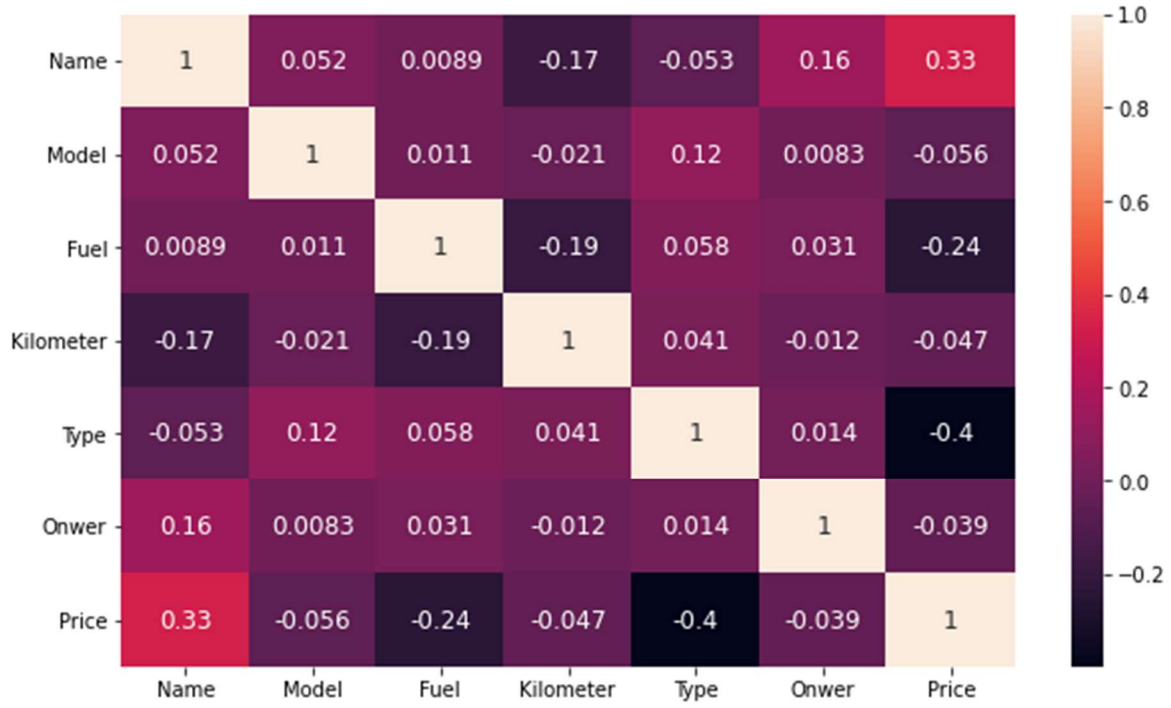
From graph we can see that the 1st onwer has the maximum cars for the sale

Similarly, done for the other variable with target variable in notebook.

Multivariate Plot:

Let's move on to analysing more than two variables now. We call it "Multivariate analysis". Now we can look at the interactions between the variables. First, let's look at heatmap plot of all pairs of attributes. This can be helpful to spot structured relationships between input variables and target variable. Let's visualize the data in this correlation matrix using a heat map.

```
: #check multicolinearity
  plt.figure(figsize=(10,6))
  sns.heatmap(data.corr(),annot=True,annot_kws={'size':12})
  plt.show()
```



From graph we can see there is no multicolinarity in the variables

Data pre-processing:

Pre-processing of this dataset includes doing analysis on the independent variables like checking for null values in each column and then replacing or filling them with supported appropriate data types like mean, mode method or Imputer methods, so that analysis and model fitting is not hindered from its way to accuracy.

Shown above are some of the representations obtained by using Pandas tools which tells about variable count for numerical columns and model values for categorical columns. Maximum and minimum values in numerical columns, along with their percentile values for median, plays an important factor in deciding which value to be chosen at priority for further exploration tasks and analysis.

Data types of different columns are used further in label processing and Label encoding scheme during model building.

There are many stages involved in data pre-processing.

Data cleaning attempts to impute missing values, removing outliers from the dataset.

Data integration integrates data from a multitude of sources into a single data warehouse.

Data transformation such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurement.

Data reduction can reduce the data size by dropping out redundant features. Feature selection and feature extraction techniques can be used.

In this dataset there are some objectives are mentioned as "--" so first lets change to particular value as per mentioned in data description. Refer below fig.

```
# Find the '--' in all the features
unkonwn={}
for i in list(data.columns):
    if (data[i]).dtype==object:
        a=np.sum(data[i]=="--")
        unkonwn[i]=a
unkonwn=pd.DataFrame.from_dict(unkonwn,orient='index')
unkonwn
```

	0
Name	0
Model	0
Fuel	2
Kilometer	0
Type	0
Onwer	3
Price	0

This we have replaced with numpy.NaN method or values. Also, in our dataset there are some spelling mistakes & duplicates are there also some unstructured data is there so we need to replace it by using pandas string method.

```
# In Kilometer column km, kms & some spelling mistecks are there so lets deal with it bu using pandas str method
data["Kilometer"]=data["Kilometer"].str[:-2]

data["Kilometer"]=data["Kilometer"].str.replace(",","")

data["Kilometer"]=data["Kilometer"].str.replace('k','')

data["Kilometer"]
```

```
0         14047
1         11510
2         50161
3         15445
```

```
# As above mentioned and from graph we see there are Rs str is present so lets deal with it by str menthod
data["Price"]=data["Price"].str.replace("₹","")

data["Price"]=data["Price"].str.replace(",","")

data['Price']=data['Price'].replace('',0)
```

As price column has null values so, lets deal with it by using mode method

```
# Dealing with null values
data['Price']=data['Price'].fillna(data['Price'].mode()[0])
```

```
# Lets change the type of column in dataset
data['Price']=data['Price'].astype(int)
data['Kilometer']=data['Kilometer'].astype(float)
```

```
# In onwer column there are spelling misteks & duplicates are there lets deal with it
data["Onwer"]=data["Onwer"].replace(['1st','first','0','1st owner'],1)

data["Onwer"]=data["Onwer"].replace(['2nd','2nd owner','second'],2)

data["Onwer"]=data["Onwer"].replace(['3rd','third','3rd owner'],3)

data["Onwer"]=data["Onwer"].replace(['4th','fourth'],4)
data["Onwer"]
```

```
0       1.0
1       2.0
```

As in our dataset have null values & objective data type so let's use Imputer techniques mean , mode method & Encoding techniques to convert data into numerical form as shown below.

Label Encoder refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning. Label encoding in python can be imported from the Sklearn library. Sklearn provides a very efficient tool for encoding. Label encoders encode labels with a value between 0 and n_classes-1.

```
# Lets frist covert categorical data(type & column) into int
label = LabelEncoder()
for i in cat_col:
    df=label.fit_transform(data[i])
    pd.Series(df)
    data[i]=df
```

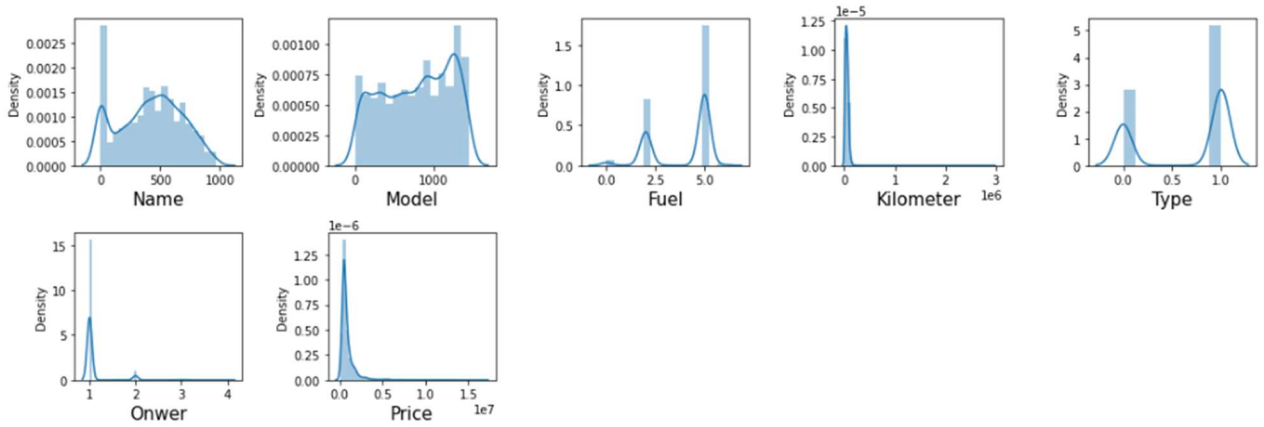
Also, as I say data pre-processing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, the greater is the reliability of the produced results.

Incomplete, noisy, and inconsistent data are the inherent nature of real-world datasets. Data pre-processing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise, and resolving inconsistencies.

Incomplete data can occur due to many reasons. Appropriate data may not be persisted due to a misunderstanding, or because of instrument defects and malfunctions.

Noisy data can occur for a number of reasons (having incorrect feature values). The instruments used for the data collection might be faulty. Data entry may contain human or instrument errors. Data transmission errors might occur as well.

```
: #Let's see the how data is distributed or Graphical analysis of all features
  plt.figure(figsize=(15,20))
  plotnumber=1
  for column in data:
      if plotnumber<=40:
          ax=plt.subplot(8,5,plotnumber)
          sns.distplot(data[column])
          plt.xlabel(column,fontsize=15)
      plotnumber+=1
  plt.tight_layout()
```

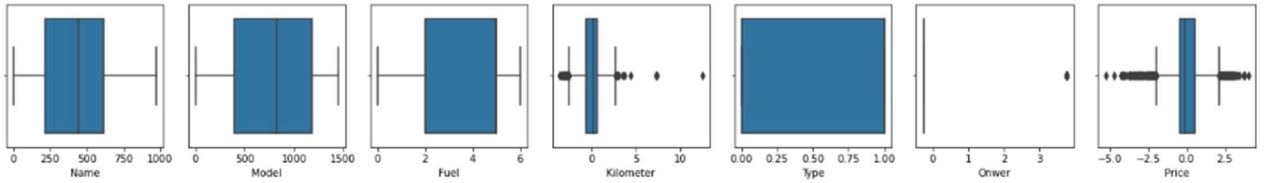


From above filling we can see the skewed data is present as the weight of values is more than +-0.5 values. So, to remove this skewness we are using Power Transformation.

```
# Using power transformation to remove skewed data
from sklearn.preprocessing import PowerTransformer
pt=PowerTransformer()
data[df1]=pt.fit_transform(data[df1].values)
```

Outliers are data points that are distant from other similar points. They may be due to variability in the measurement or may indicate experimental errors. If possible, outliers should be excluded from the data set. However, detecting that anomalous instance might be very difficult, and is not always possible.

```
#Let's see the outliers in the dataset by using box plot or Graphical analysis of all features
myFig=plt.figure(figsize=(20,20))
plotnumber=1
for column in data:
    if plotnumber<=64:
        ax=plt.subplot(8,8,plotnumber)
        sns.boxplot(data[column])
        plt.xlabel(column,fontsize=10)
    plotnumber+=1
plt.tight_layout()
```



From graph we can see the outliers are present columns so lets deal with it by using Zscore

Methods we used to remove outliers from our dataset is:

Z-score — Call scipy.stats.zscore() with the given data-frame as its argument to get a numpy array containing the z-score of each value in a dataframe. Call numpy.abs() with the previous result to convert each element in the dataframe to its absolute value. Use the syntax (array < 3).all(axis=1) with array as the previous result to create a boolean array.

```
# from above graph we see there is outliers in featurs Let's remove outliers from above columns by using Zscore
z_score=zscore(data[df1])
abs_z_score=np.abs(z_score)
filtering_entry=(abs_z_score<3).all(axis=1)
data=data[filtering_entry]
```

Correlations among variables:

Heatmap was plotted for variables with correlation coefficient. The Variance Inflation Factor (VIF) measures the severity of multicollinearity in regression analysis. It is a statistical concept that indicates the increase in the variance of a regression coefficient as a result of collinearity. Variance inflation factor (VIF) is used to detect the severity of multicollinearity in the ordinary least square (OLS) regression analysis. Multicollinearity inflates the variance and type II error. It makes the coefficient of a variable consistent but unreliable. VIF measures the number of inflated variances caused by multicollinearity. Checking correlation between dependent and independent variables.

```
#check multicolinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif"]=[variance_inflation_factor(x_scaled,i) for i in range(x_scaled.shape[1])]
vif["featurs"]=x.columns
print(vif)
```

```
        vif    featurs
0  1.231508       Name
1  1.022937      Model
2  1.154433       Fuel
3  1.378607  Kilometer
4  1.038745       Type
5  1.029229      Onwer
```

No colinarity in the variables

Building machine learning models:

For building machine learning models there are several models present inside the Sklearn module.

Sklearn provides two types of models i.e., regression and classification. Our datasets contain continuous type target variable i.e**. Price.** So, this kind of problem, we use regression models. But before fitting our dataset to its model first we have to separate the predictor variable and the target variable, then scaled the independent variables or predictors by using

StandardScaler method and then we pass this variable to the train_test_split method to create a random test and train subset.

What is train_test_split, it is a function in sklearn model selection for splitting data arrays into two subsets for training data and testing data. With this function, you don't need to divide the dataset manually. By default, sklearn train_test_split will make random partitions for the two subsets.

However, you can also specify a random state for the operation. It gives us four outputs x_train, x_test, y_train and y_test. The x_train and x_test contains the training and testing predictor variables while y_train and y_test contains the training and testing target variable. After performing train_test_split we have to choose the models to pass the training variable.

We can build as many models as we want to compare the accuracy given by these models and to select the best model among them.

I have selected 5 models for the problem statement that is prediction of Car Price:

1) **LinearRegression from sklearn.linear_model:** Linear regression models are most preferably used with the least-squares approach, where the implementation might require other ways by minimising the deviations and the cost functions, for instance. The general linear models include a response variable that is a vector in nature and not directly scalar. The conditional linearity is still presumed positive over the modelling process. They vary over a large scale, but they are better described as the skewed distribution, which is related to the log-normal distribution.

```
# Model no.1
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)

print_score(lr,x_train,x_test,y_train,y_test,train=True)
print_score(lr,x_train,x_test,y_train,y_test,train=False)
model_accuracy(lr)
```

```
Train Report: 0.5193460077049958
Test Report: 0.5114747301319231
RMSE: 0.6452877894108415
MAE: 0.4997817703646926
MSE: 0.41639633116273056
Accuracy: 51.54 %
Standard Deviation: 2.34 %
```

2) **Lasso regression from sklearn.model:** Lasso regression is a method we can use to fit a regression model when multicollinearity is present in the data.

   In a nutshell, least squares regression tries to find coefficient estimates that minimize the sum of squared residuals (RSS):

   RSS = $\Sigma(y_i - \hat{y}_i)^2$
   where:

Σ: A greek symbol that means sum
yi: The actual response value for the ith observation
ŷi: The predicted response value based on the multiple linear regression model

Conversely, lasso regression seeks to minimize the following:

RSS + λΣ|βj|
where j ranges from 1 to p predictor variables and λ ≥ 0.

This second term in the equation is known as a shrinkage penalty. In lasso regression, we select a value for λ that produces the lowest possible test MSE (mean squared error).

```
# Model no.2
from sklearn.linear_model import LinearRegression, Lasso, LassoCV

lcv=LassoCV(alphas=None,max_iter=10000,normalize=True)
lcv.fit(x_train,y_train)
alpha=lcv.alpha_
print(alpha)
Lasso_reg=Lasso(alpha).fit(x_train,y_train)

print_score(Lasso_reg,x_train,x_test,y_train,y_test,train=True)
print_score(Lasso_reg,x_train,x_test,y_train,y_test,train=False)
model_accuracy(Lasso_reg)
```

```
8.884455708620262e-06
Train Report: 0.519346007131841
Test Report: 0.5114751053286833
RMSE: 0.6452875416141057
MAE: 0.49977972080273936
MSE: 0.4163960113623762
Accuracy: 51.54 %
Standard Deviation: 2.34 %
```

3) **DecisionTreeRegressor from sklearn.tree:** Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. The two main entities of a tree are decision nodes, where the data is split and leaves, where we get the outcome.

```
#Model no.5
from sklearn.tree import DecisionTreeRegressor

dt=DecisionTreeRegressor()

dt.fit(x_train,y_train)

print_score(dt,x_train,x_test,y_train,y_test,train=True)
print_score(dt,x_train,x_test,y_train,y_test,train=False)
model_accuracy(dt)
```

```
Train Report: 0.9999545796434472
Test Report: 0.6601058719637307
RMSE: 0.5382471778757553
MAE: 0.3236724154000528
MSE: 0.2897100244912149
Accuracy: 62.60 %
Standard Deviation: 5.12 %
```

4) **RandomForestRegressor from sklearn.ensemble:** As we know that a forest is made up of trees and more trees means more robust forest. Similarly, a random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

```
# Model no.3
from sklearn.ensemble import RandomForestRegressor

rand_regressor= RandomForestRegressor()
rand_regressor.fit(x_train,y_train)

print_score(rand_regressor,x_train,x_test,y_train,y_test,train=True)
print_score(rand_regressor,x_train,x_test,y_train,y_test,train=False)
model_accuracy(rand_regressor)
```

```
Train Report: 0.9711304747734515
Test Report: 0.8253203112040499
RMSE: 0.385861221919339
MAE: 0.2511521516034769
MSE: 0.1488888825810854
Accuracy: 77.49 %
Standard Deviation: 3.71 %
```

5) **XGBRegressor from XGBoost:** XGBoost is short for "eXtreme Gradient Boosting." The "eXtreme" refers to speed enhancements such as parallel computing and cache awareness that makes XGBoost approximately 10 times faster than traditional Gradient Boosting. In addition, XGBoost includes a unique split-finding algorithm to optimise trees, along with built-in regularisation that reduces over-fitting. Generally speaking, XGBoost is a faster, more accurate version of Gradient Boosting.

```
# Model no.3
from xgboost import XGBRegressor

xgb=XGBRegressor()
xgb.fit(x_train,y_train)

print_score(xgb,x_train,x_test,y_train,y_test,train=True)
print_score(xgb,x_train,x_test,y_train,y_test,train=False)
model_accuracy(xgb)
```

```
Train Report: 0.9784533583199685
Test Report: 0.819835071320794
RMSE: 0.3918727436318222
MAE: 0.26216878094627505
MSE: 0.15356424720153183
Accuracy: 79.55 %
Standard Deviation: 3.26 %
```

6) **HistGradientBoostingRegressor:** The number of tree that are built at each iteration. For regressors, this is always 1. train_score_ndarray, shape (n_iter_+1,) The scores at each iteration on the training data. The first entry is the score of the ensemble before the first iteration. Scores are computed according to the scoring parameter.

```
#Model no.4
from sklearn.ensemble import HistGradientBoostingRegressor

gbdt=HistGradientBoostingRegressor()

gbdt.fit(x_train,y_train)

print_score(gbdt,x_train,x_test,y_train,y_test,train=True)
print_score(gbdt,x_train,x_test,y_train,y_test,train=False)
model_accuracy(gbdt)
```

```
Train Report: 0.8996621377069951
Test Report: 0.8130204258574354
RMSE: 0.39921514952939363
MAE: 0.2781181290296519
MSE: 0.15937273561377613
Accuracy: 78.61 %
Standard Deviation: 3.62 %
```

As we have both the train and test dataset, we have used train dataset to build the model and based on this model we are going to predict the sales price for this data so, we are doing all the EDA steps as per train data and predicting the sales price.

We got **our** best model looking at accuracy, is **LassoRegression** with Kfold cross validation method with the accuracy score of 51.54 % With RMSE: 0.6, MAE: 0.4, & MSE: 0.4 % error.

Understand what does MSE, MAE & RMSE do:

Mean squared error	$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}$
Mean absolute error	$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e$

MSE: **Mean squared error**

In machine learning, the mean squared error (MSE) is used to evaluate the performance of a regression model. In regression models, the RMSE is used as a metric to measure model performance and the MSE score is used to evaluate the performance. The MSE score is used to evaluate the performance of a machine learning model while working on regression

problems. When the distance is higher it represents a high error rate and when the distance is lower than you are near to the best fit line.

MAE: Mean Absolute Error
Mean Absolute Error metric is to subtract our predicted value and actual value at each time point to obtain the absolute value, and then average it out.

RMSE: Root Mean Squared Error

Root Mean Squared Error (RMSE) is a common metric for assessing the performance of regression machine learning models. It is often used to provide a metric that is related to the unit being measured.

Squared error, also known as L2 loss, is a row-level error calculation where the difference between the prediction and the actual is squared. RMSE is the aggregated mean and subsequent square root of these errors, which helps us understand the model performance over the whole dataset.

A benefit of using RMSE is that the metric it produces is on the same scale as the unit being predicted. For example, calculating RMSE for a house price prediction model would give the error in terms of house price, which can help end users easily understand model performance.

Cross validation:

Models are trained with a 5-fold cross validation. A technique that takes the entirety of your training data, randomly splits it into train and validation data sets over 5 iterations.

You end up with 5 different training and validation data sets to build and test your models. It's a good way to counter overfitting.

More generally, cross validation of this kind is known as k-fold cross validation.

Hyper parameter tuning:

Hyper parameter optimisation in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyper parameters, in contrast to model parameters, are set by the machine learning engineer before training.

The number of trees in a random forest is a hyper parameter while the weights in a neural network are model parameters learned during training. I like to think of hyper parameters as the model settings to be tuned so that the model can optimally solve the machine learning problem.

We will use RandomizedSearchCV for the hyper parameter tuning.

RandomizedSearchCV :

Randomized search on hyper parameters. RandomizedSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba",

“decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

But in this our problem after using the hyper tuning parameter there is no change in accuracy and error so no need to do the same.

Remarks:

In this project we build the regression model that can predict car price of house. The challenge behind predicting models is scraping the data from different website and EDA.

We have gone through how to implement the entire machine learning pipeline, and we have an intuitive understanding of machine learning algorithms. The larger the dataset gets, the more complex each of the mentioned steps gets. Therefore, using this as a base will help while you build your knowledge of machine learning pipelines.

This Paper has presented a supervised housing sales price learning model which used machine learning algorithms to predict the price. We used different machine learning algorithm to check the accuracy of price prediction.

References:

- Regression Model
- Hyper-parameter Tuning
- Cross Validation
- Getting started with XGBoost
- scikit

In a nutshell….

Exploring and knowing your datasets is a very essential step. It not only helps in finding anomalies, uniqueness and pattern in the dataset but also helps us in building better hypothesis functions. If you wish to see the entire code, here Link is the to my Jupiter notebook