FLIP ROBO

PROJECT REPORT

ON

Old Cars Selling Price Prediction

Submitted by: M Ashok Kumar

ACKNOWLEDGMENT

Approximately 40 million used vehicles are sold each year. Effective pricing strategies can help any company to efficiently sell its products in a competitive market and make a profit this blog,

The purpose of this project is to conduct a mini-project that encompasses the bulk of the data science process — from data collection (web-scraping: Beautiful Soup, Python), data cleaning, exploratory data analysis, model training, and testing stage.

The source of data comes from cars24, and www.cardekho.com , which are online car sales portals in India.

I will be **analysing the Old Cars Selling Price prediction using a Machine Learning dataset** using essential exploratory data analysis techniques and also, and I will be performing some data visualizations to better understand our data.

By doing data preprocessing, data analysis, feature selection, and many other techniques we built our cool and fancy machine learning model. And at the end, we applied many ml algorithms to get the very good accuracy of our model.

Many thanks to Fliprobo Technology for providing me with this project to understand the Real-Time Field work present in Data Science Industry.

I am very thankful to my friends and family who helped me through this study. So without any further due.

ABSTRACT

The prices of new cars in the industry are fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So, customers buying a new car can be assured of the money they invest to be worthy. But due to the increased price of new cars and the incapability of customers to buy new cars due to the lack of funds, used car sales are on a global increase (Pal, Arora and Palakurthy, 2018). There is a need for a used car price prediction system to effectively determine the worthiness of the car using a variety of features. Even though there are websites that offer this service, their prediction method may not be the best. Besides, different models and systems may contribute to predicting the power of a used car's actual market value. It is important to know their actual market value while both buying and selling.

TAKEAWAYS FROM THE BLOG

In this article, we do prediction using machine learning which leads to the below takeaways:

1. **Web Scraping:** Scraping data from websites like cars24, cardekho.com and olx.
2. **EDA:** Learn the complete process of EDA
3. **Data analysis:** Learn to withdraw some insights from the dataset both mathematically and visualize it.
4. **Data visualization:** Visualizing the data to get better insight from it.
5. **Feature engineering:** We will also see what kind of stuff we can do in the feature engineering part.
6. Eliminating features that had an insignificant effect on the response variable by evaluating the p-values and R^2 value of the mode

PROBLEM STATEMENT:

With the covid 19 impact in the market, we have seen a 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 the clients works with small traders, who sell used cars. With the change in the market due to covid 19 impact, THE 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.

Model Building Phase After collecting/scraping the data, we have around 5000 rows and 9 columns. We need to build a machine learning model. Before model building, we will be doing data pre-processing steps. We will try different models with different hyper parameters and select the best model.

ABOUT THE DATASET

About the data:

1. Number of features in dataset: 9

2. Number of data points in dataset: 5058

We have scraped price of 5058 old cars from Cars24 and cardekho.com This problem involves predicting the Selling prices of the old cars which are continuous and real-valued outputs. Thus, this is a **Regression Problem.**

Features:

Here's a brief version of what features is in the data description file:

1. **Brand Name:** Model and brand name of the car.

2. **Year**: Year of manufacturing/ model launch year

3. **Car_variant**: Model type

4. **Selling Price:** The price on which the car is available on the website

5. **Kilometers_Driven:** How many km the car is driven till now

6. **Fuel_Type:** if the model run by petrol or diesel

7. **Transmission:** If the model is run manually or automatic

8. **Owner_Type**: How many owners are there or what source of car

9. **Location:** Which city of India the car is available importing Important Libraries:

Importing Important libraries:

We need some libraries to be imported to work upon the dataset, we would import the dataset by using pandas' read_csv method.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso,Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error
from scipy.stats import skew,norm

import warnings
warnings.filterwarnings("ignore")
```

Loading Data Set into a variable:

Here I am loading the dataset into the variable cars_df.

```
cars_df = pd.read_csv("Old_Car_details.csv")
cars_df
```

	Brand Name	Year	Car_variant	Selling Price	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Location
0	Mercedes Benz E Class	2013	E 250 CDI AVANTGARDE	1455599	42881	Diesel	Automatic	2nd Owner	New Delhi
1	Honda City	2014	V MT PETROL	571499	51778	Petrol	Manual	1st Owner	New Delhi
2	Volkswagen Polo	2016	COMFORTLINE 1.2L PETROL	420399	59206	Petrol	Manual	2nd Owner	New Delhi
3	Maruti Wagon R 1.0	2014	LXI	333199	62798	Petrol	Manual	1st Owner	New Delhi
4	Honda City	2020	ZX CVT	1456371	9928	Petrol	Automatic	1st Owner	New Delhi
...	...	...	...	...	...	...	...	...	...
5053	Honda Civic	2019	VX Diesel	1450000	48523	Diesel	Manual	NaN	New Delhi
5054	Datsun RediGO	2017	1.0 S	220000	34683	Petrol	Manual	NaN	New Delhi
5055	Toyota Innova Crysta	2020	2.4 GX 7 STR AT	1939000	76317	Diesel	Automatic	NaN	New Delhi
5056	Maruti Swift	2017	LDI Optional	419000	82011	Diesel	Manual	NaN	New Delhi
5057	Maruti Baleno	2019	Delta	690000	35992	Petrol	Manual	NaN	New Delhi

5058 rows × 9 columns

Selling Price is the target variable.

Exploratory Data Analysis:

E**xploratory Data Analysis** refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. We performed some bi-variate analysis on the data to get a better overview of the data and to find outliers in our data-set. Outliers can occur due to some kind of errors while collecting the data and need to be removed so that it doesn't affect the performance of our model.

Checking the shape of the dataset:

```
cars_df.shape
```

```
(5058, 9)
```

```
cars_df.columns
```

```
Index(['Brand Name', 'Year', 'Car_variant', 'Selling Price',
       'Kilometers_Driven', 'Fuel_Type', 'Transmission', 'Owner_Type',
       'Location'],
      dtype='object')
```

We can see that the dataset has 5058 rows and 9 columns..

Getting detailed information about both the datasets:

We have two datasets.

The dataset has 5058 observations and 9 columns including the target variable. The target variable is Selling price which is of integer data type.

After applying the info() function to the datasets we get to know the data types of all the features and missing values of both the datasets.

```
1  cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5058 entries, 0 to 5057
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Brand Name         5058 non-null   object
 1   Year               5058 non-null   int64 
 2   Car_variant        5058 non-null   object
 3   Selling Price      5058 non-null   int64 
 4   Kilometers_Driven  5058 non-null   int64 
 5   Fuel_Type          5058 non-null   object
 6   Transmission       5002 non-null   object
 7   Owner_Type         4898 non-null   object
 8   Location           5058 non-null   object
dtypes: int64(3), object(6)
memory usage: 355.8+ KB
```

We concluded that we have 3 numerical and rest 6 categorical data types features in the datasets. Also, there are so many missing values in the Transmission and Owner type column.

Separating Brand and Model name From Brand name column:

```
1  cars_df['Brand']=cars_df['Brand Name'].str.extract(r'(\w+)\b')
2  cars_df.head()
```

	Brand Name	Year	Car_variant	Selling Price	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Location	Brand
0	Mercedes Benz E Class	2013	E 250 CDI AVANTGARDE	1455599	42881	Diesel	Automatic	2nd Owner	New Delhi	Mercedes
1	Honda City	2014	V MT PETROL	571499	51778	Petrol	Manual	1st Owner	New Delhi	Honda
2	Volkswagen Polo	2016	COMFORTLINE 1.2L PETROL	420399	59206	Petrol	Manual	2nd Owner	New Delhi	Volkswagen
3	Maruti Wagon R 1.0	2014	LXI	333199	62798	Petrol	Manual	1st Owner	New Delhi	Maruti
4	Honda City	2020	ZX CVT	1456371	9928	Petrol	Automatic	1st Owner	New Delhi	Honda

```
model=cars_df['Brand Name'].str.split().str[1:]
cars_df['Model']=[' '.join(map(str, l)) for l in model]
cars_df.head()
```

	Brand Name	Year	Car_variant	Selling Price	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Location	Brand	Model
0	Mercedes Benz E Class	2013	E 250 CDI AVANTGARDE	1455599	42881	Diesel	Automatic	2nd Owner	New Delhi	Mercedes	Benz E Class
1	Honda City	2014	V MT PETROL	571499	51778	Petrol	Manual	1st Owner	New Delhi	Honda	City
2	Volkswagen Polo	2016	COMFORTLINE 1.2L PETROL	420399	59206	Petrol	Manual	2nd Owner	New Delhi	Volkswagen	Polo
3	Maruti Wagon R 1.0	2014	LXI	333199	62798	Petrol	Manual	1st Owner	New Delhi	Maruti	Wagon R 1.0
4	Honda City	2020	ZX CVT	1456371	9928	Petrol	Automatic	1st Owner	New Delhi	Honda	City

Now we can drop the brand name column as we have separate columns for brand and model

```
# Since we have separated the Brand and Model name from Brand name column, we will drop the Brand Name column
cars_df.drop("Brand Name",axis =1,inplace = True)
cars_df
```

	Year	Car_variant	Selling Price	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Location	Brand	Model
0	2013	E 250 CDI AVANTGARDE	1455599	42881	Diesel	Automatic	2nd Owner	New Delhi	Mercedes	Benz E Class
1	2014	V MT PETROL	571499	51778	Petrol	Manual	1st Owner	New Delhi	Honda	City
2	2016	COMFORTLINE 1.2L PETROL	420399	59206	Petrol	Manual	2nd Owner	New Delhi	Volkswagen	Polo
3	2014	LXI	333199	62798	Petrol	Manual	1st Owner	New Delhi	Maruti	Wagon R 1.0
4	2020	ZX CVT	1456371	9928	Petrol	Automatic	1st Owner	New Delhi	Honda	City

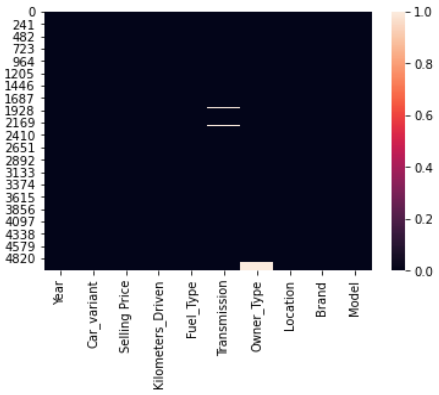
Checking the Null Values:

```
cars_df.isnull().sum()
```

```
Year                 0
Car_variant          0
Selling Price        0
Kilometers_Driven    0
Fuel_Type            0
Transmission        56
Owner_Type         160
Location             0
Brand                0
Model                0
dtype: int64
```

```
sns.heatmap(cars_df.isnull())
```

```
<AxesSubplot:>
```



Treating the Missing values in the dataset:

We have null values in Transmission and Owner Type and both are categorical in nature. So we will apply Simple Imputer (most_frequent) strategy

```
from sklearn.impute import SimpleImputer
imp= SimpleImputer(strategy="most_frequent")

cars_df["Transmission"]= imp.fit_transform(cars_df["Transmission"].values.reshape(-1,1))
cars_df["Owner_Type"]= imp.fit_transform(cars_df["Owner_Type"].values.reshape(-1,1))
```

Checking the null values again

```
#Again checking the null values
cars_df.isnull().sum()
```

```
Year                0
Car_variant         0
Selling Price       0
Kilometers_Driven   0
Fuel_Type           0
Transmission        0
Owner_Type          0
Location            0
Brand               0
Model               0
dtype: int64
```

Now there is no null values in both the datasets.

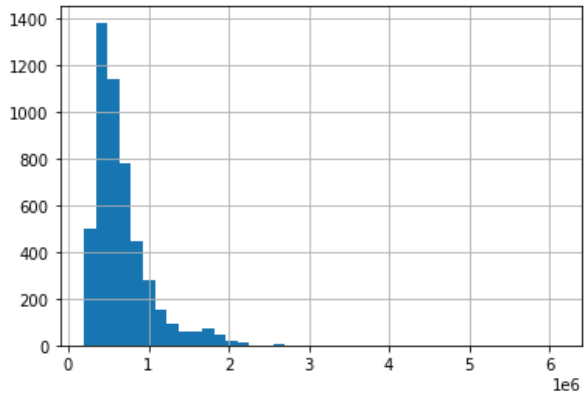
Univariate Analysis:

Uni means one, so in other words, the data has only one variable. Univariate data requires analysing each variable separately. It doesn't deal with causes or relationships (unlike regression) and its major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.

Analyzing Target variable:

```
#histogram
cars_df['Selling Price'].hist(bins = 40)
```

```
<AxesSubplot:>
```




```
#descriptive statistics summary
cars_df['Selling Price'].describe()
```

```
count    5.058000e+03
mean     6.635944e+05
std      3.782556e+05
min      1.923990e+05
25%      4.233240e+05
50%      5.608790e+05
75%      7.771740e+05
max      6.100000e+06
Name: Selling Price, dtype: float64
```

We can clearly see that the target variable has a normal distribution that is skewed towards the right. Now let's calculate the Skewness and Kurtosis:

```
#skewness & kurtosis
print("Skewness: %f" % cars_df['Selling Price'].skew())
print("Kurtosis: %f" % cars_df['Selling Price'].kurt())
```

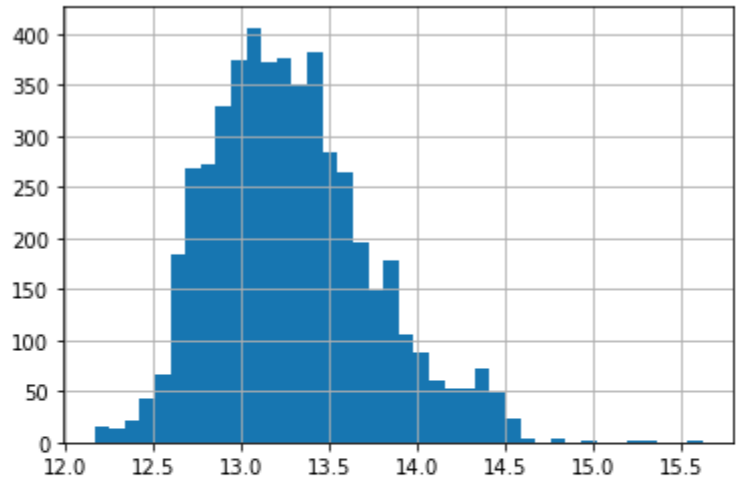
```
Skewness: 3.277941
Kurtosis: 25.861284
```

As we saw before, the target variable "Selling Price" is not uniformly distributed and it's skewed towards the right. Therefore**, we will try to use the log transformation to remove the skewness**.

```
#log transform the target
cars_df["Selling Price"] = np.log1p(cars_df["Selling Price"])

#histogram
cars_df['Selling Price'].hist(bins = 40)
```

```
<AxesSubplot:>
```



Now the data in target column is uniformly distributed.

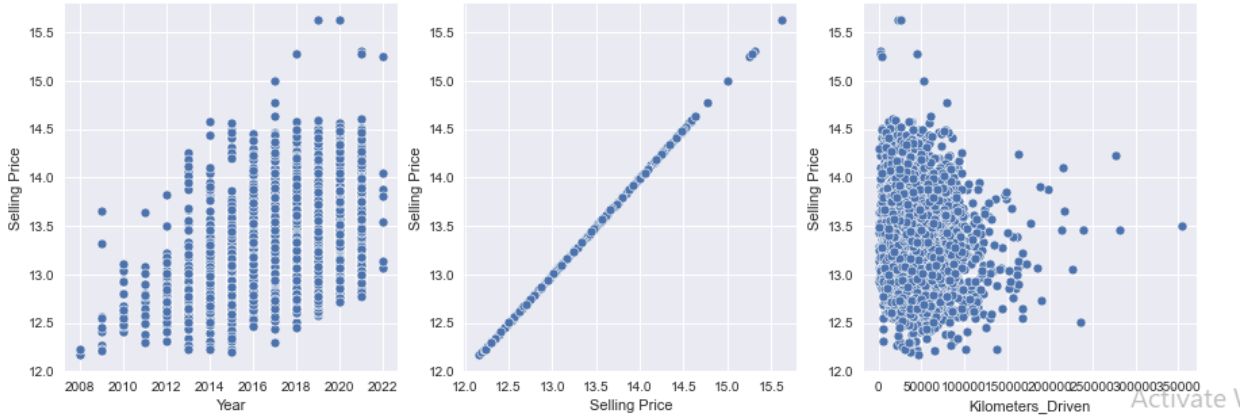
```
#skewness & kurtosis
print("Skewness: %f" % cars_df['Selling Price'].skew())
print("Kurtosis: %f" % cars_df['Selling Price'].kurt())
```

```
Skewness: 0.602862
Kurtosis: 0.395709
```

Now both the skewness and kurtosis are removed in the target variable.This looks almost normal distribution with a **Skew of 0.602862.**

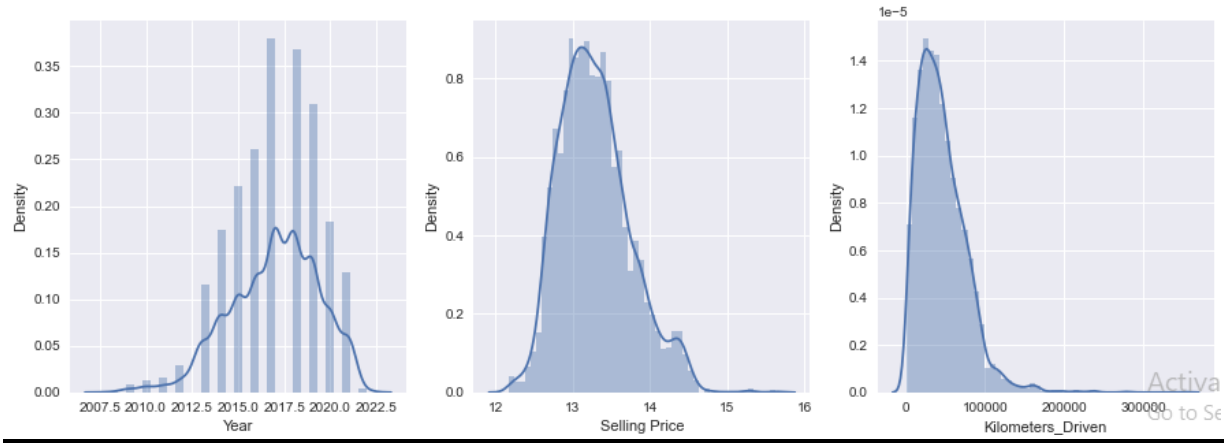
Analysing Numerical columns:

```
plt.figure(figsize=[15,5])
num_var = cars_df.select_dtypes(exclude= ["O"]).columns
plt.style.use("seaborn")
for i,col in enumerate(num_var):
    plt.subplot(1,3,i+1)
    sns.scatterplot(cars_df[col],cars_df['Selling Price'])
plt.show()
```



Checking skewness using distplot

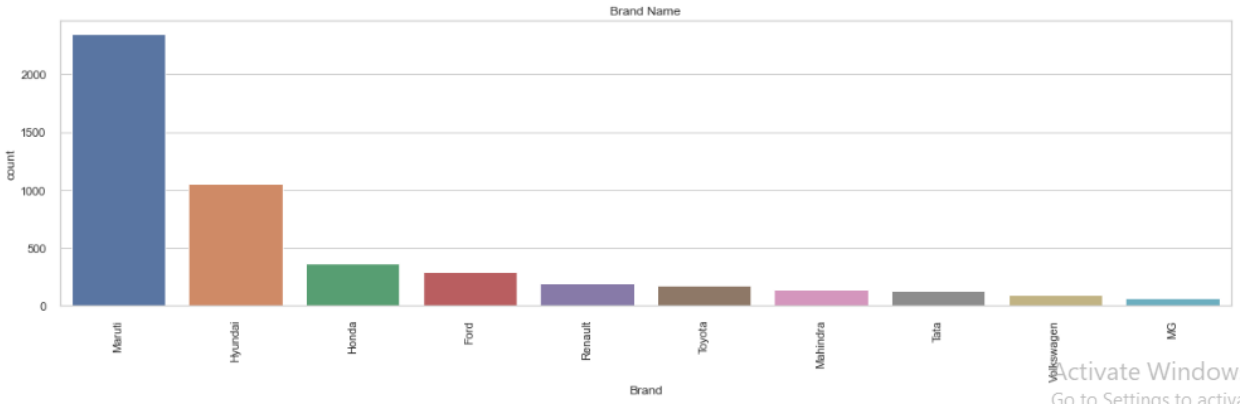
```
plt.figure(figsize=[15,5])
num_var = cars_df.select_dtypes(exclude= ["O"]).columns
plt.style.use("seaborn")
for i,col in enumerate(num_var):
    plt.subplot(1,3,i+1)
    sns.distplot(cars_df[col])
plt.show()
```



Here in all the numerical columns we can see through scatter plot that many numerical features are uniformly distributed.

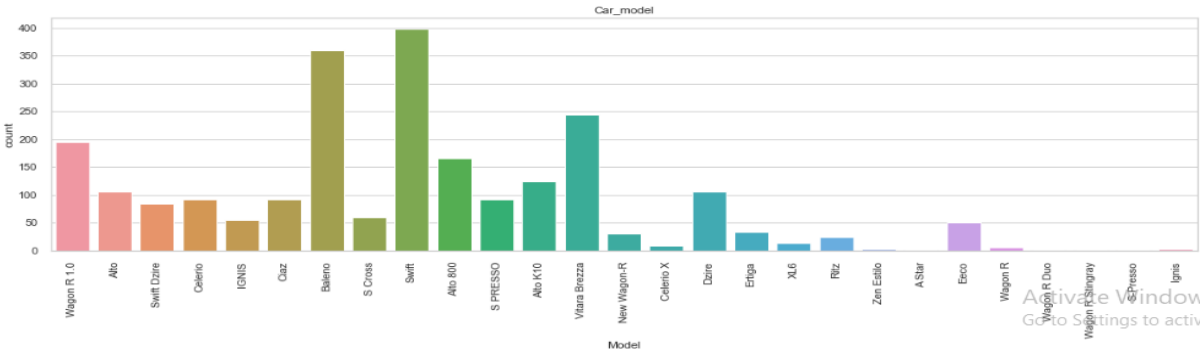
Analysing Categorical Columns:

```
# Checking Top 10 selling brands of cars
sns.set(style="whitegrid")
plt.figure(figsize=(20,5))
sns.countplot(cars_df['Brand'],order=cars_df['Brand'].value_counts().iloc[:10].index)
plt.title("Brand Name")
plt.xticks(rotation=90)
plt.show()
```



We can see from above count plot, Maruti is the highest selling brand followed by Hyundai. This may be because of their genuine price with genuine features.

```
sns.set(style="whitegrid")
plt.figure(figsize=(20,5))
sns.countplot(cars_df[cars_df['Brand']=="Maruti"]['Model'])
plt.title("Car_model")
plt.xticks(rotation=90)
plt.show()
```



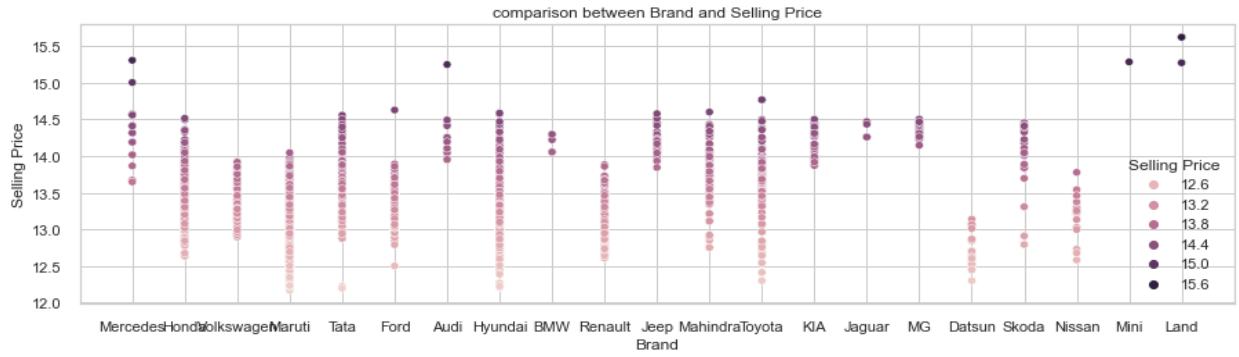
From the above count plot we can see Maruti Swift model is highest selling model, folled by Baleno And Brezza.

Bivariate Analysis:

Bivariate analysis is finding some kind of empirical relationship between two variables. Specifically, the dependent vs independent Variables

```
plt.figure(figsize=[15,4])
plt.title("comparison between Brand and Selling Price")
sns.scatterplot(cars_df['Brand'],cars_df['Selling Price'],hue=cars_df["Selling Price"])
```

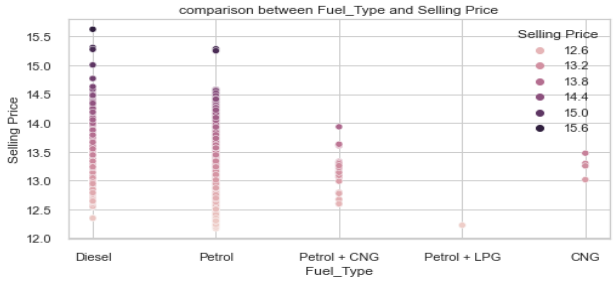
```
<AxesSubplot:title={'center':'comparison between Brand and Selling Price'}, xlabel='Brand', ylabel='Selling Price'>
```



Here we can Land Rover, Mercedes, Audi have highest selling Price as they are top brands .

```
plt.figure(figsize=[8,4])
plt.title("comparison between Fuel_Type and Selling Price")
sns.scatterplot(cars_df['Fuel_Type'],cars_df['Selling Price'],hue=cars_df["Selling Price"])
```

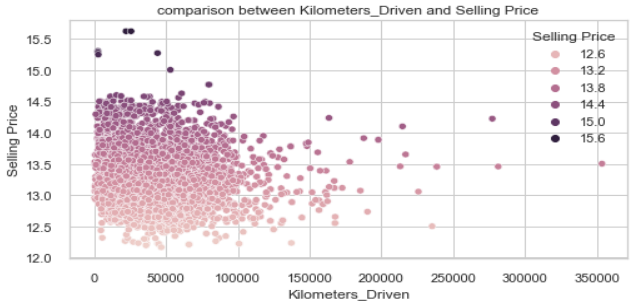
```
<AxesSubplot:title={'center':'comparison between Fuel_Type and Selling Price'}, xlabel='Fuel_Type', ylabel='Selling Price'>
```



We can see from the above plot that Diesel cars are costly .They have highest selling price, followed by petrol cars. Petrol+LPG have lowest selling price.

```
plt.figure(figsize=[8,4])
plt.title("comparison between Kilometers_Driven and Selling Price")
sns.scatterplot(cars_df['Kilometers_Driven'],cars_df['Selling Price'],hue=cars_df["Selling Price"])
```

```
<AxesSubplot:title={'center':'comparison between Kilometers_Driven and Selling Price'}, xlabel='Kilometers_Driven', ylabel='Sel
ling Price'>
```



Selling Price does not depend much on Kilometer_driven. It is mainly affected by Model of the cars and the fuel type together with Kilometer Driven.

Categorical Columns – Value Counts

```python
# For train data
cat_col_tr = cars_df.select_dtypes(include ="object")
for i in cat_col_tr:
    print(i)
    print(cat_col_tr[i].value_counts())
    print('\n')
```

```
Car_variant
VXI                        491
LXI                        244
VDI                        221
DELTA 1.2 K12              191
SPORTZ 1.2 KAPPA VTVT      105
                          ...
RXZ PACK DIESEL 110          1
PURE MT                      1
ACTIVE 1.0 TSI               1
W10 2WD                      1
MAGNA 1.2 AT                 1
Name: Car_variant, Length: 752, dtype: int64


Fuel_Type
Petrol          3437
Diesel          1554
Petrol + CNG      62
CNG                4
Petrol + LPG       1
Name: Fuel_Type, dtype: int64
```

```
Transmission
Manual       4209
Automatic     793
Name: Transmission, dtype: int64


Owner_Type
1st Owner    3882
2nd Owner     922
3rd Owner      91
4th Owner       3
Name: Owner_Type, dtype: int64


Location
New Delhi    866
Noida        706
Gurgaon      706
Mumbai       601
Chennai      552
Pune         439
Bengaluru    387
Ahmedabad    383
Hyderabad    244
Kolkata      174
Name: Location, dtype: int64
```

```
Brand
Maruti        2346
Hyundai       1052
Honda          365
Ford           289
Renault        190
Toyota         175
Mahindra       139
Tata           123
Volkswagen      90
MG              64
Jeep            49
KIA             41
Audi            32
Datsun          25
Mercedes        23
Skoda           20
Nissan          17
BMW              9
Jaguar           5
Land             3
Mini             1
Name: Brand, dtype: int64


Model
Swift                398
Baleno               359
Ecosport             255
Grand i10            251
Vitara Brezza        244
                    ...
Benz B Class           1
A Star                 1
Wagon R Stingray       1
Bolero                 1
KUSHAQ                 1
Name: Model, Length: 145, dtype: int64
```

Converting categorical columns to numerical columns

We will convert categorical columns into numerical columns using label encoder for further analysis.

```python
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
cars_df=cars_df.apply(LabelEncoder().fit_transform)
print(cars_df.head())
```

```
   Year  Car_variant  Selling Price  Kilometers_Driven  Fuel_Type  \
0     5          307           2779               2054          1
1     6          567           1396               2443          2
2     8          275            649               2693          2
3     6          390            246               2806          2
4    12          713           2780                320          2

   Transmission  Owner_Type  Location  Brand  Model
0             0           1         7     13     19
1             1           0         7      4     30
2             1           1         7     20     90
3             1           0         7     12    129
4             0           0         7      4     30
```

Checking for Correlation with Output Features:

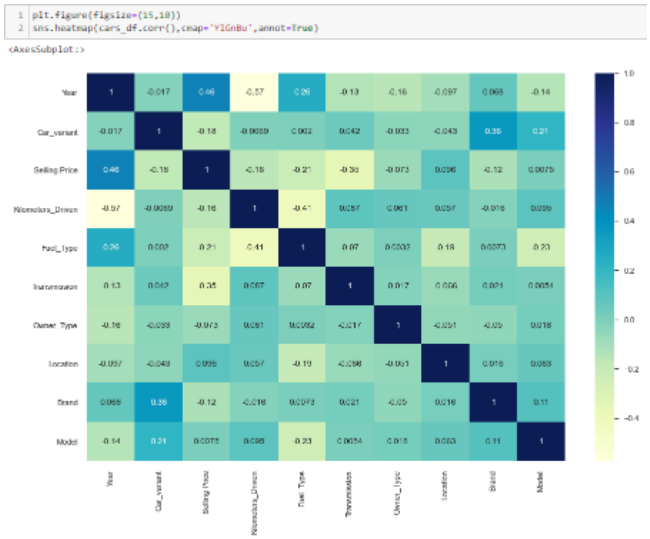
After this, we found the most important features relative to the target by building a correlation matrix. **A correlation matrix** is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The correlation coefficient has values **between -1 to 1.**

Correlations are very useful in many applications, especially when conducting regression analysis.

Multicollinearity:

Multicollinearity is **a statistical concept where several independent variables in a model are correlated**. Two variables are considered to be perfectly collinear if their correlation coefficient is +/- 1.0. Multicollinearity among independent variables will result in less reliable statistical inferences.

Let’s check the correlation and multicollinearity through correlation heat map.



Observation:

1. Year of manufacturing, Location, Brand, and Model is positively correlated with the selling price. It means more latest is the model and more recently the model was manufactured, more is the price.
2. Other features are negatively correlated with the target feature.
3. There is not much multicollinearity among the independent features.

Separating Independent and Dependent (target) features from Train Data:

```
X = cars_df.drop('Selling Price', axis=1)
y = cars_df['Selling Price'].values
```

Check for Outliers:

An outlier is **a data point that is noticeably different from the rest**. They represent errors in measurement, bad data collection, or simply show variables not considered when collecting the data. **A value that "lies outside" (is much smaller or larger than) most of the other values in a set of data**.

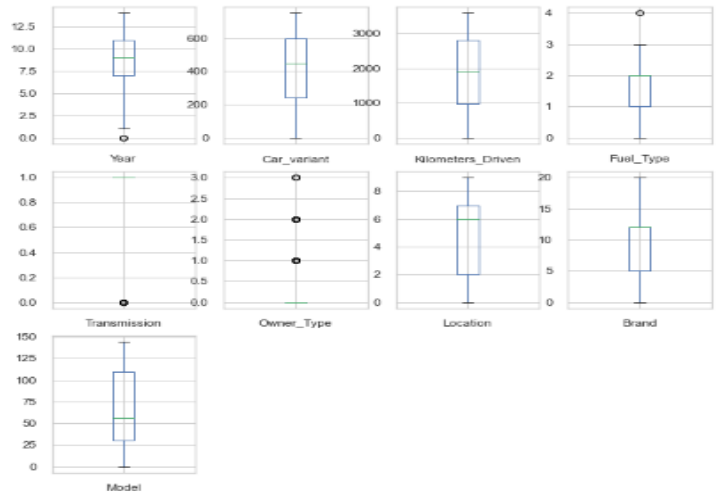
Box Plot

This is the visual representation of the depicting groups of numerical data through their quartiles. Boxplot **is also used for detecting the outlier in the data set.**

I used a box plot in this dataset because It captures the summary of the data efficiently with a simple box and whiskers and allows me to compare easily across groups.

```
X.plot(kind='box',subplots=True,layout=(3,4),figsize=(10,10))
```

```
Year                AxesSubplot(0.125,0.657941;0.168478x0.222059)
Car_variant         AxesSubplot(0.327174,0.657941;0.168478x0.222059)
Kilometers_Driven   AxesSubplot(0.529348,0.657941;0.168478x0.222059)
Fuel_Type           AxesSubplot(0.731522,0.657941;0.168478x0.222059)
Transmission           AxesSubplot(0.125,0.391471;0.168478x0.222059)
Owner_Type          AxesSubplot(0.327174,0.391471;0.168478x0.222059)
Location            AxesSubplot(0.529348,0.391471;0.168478x0.222059)
Brand               AxesSubplot(0.731522,0.391471;0.168478x0.222059)
Model                  AxesSubplot(0.125,0.125;0.168478x0.222059)
dtype: object
```



There are not many outliers in the columns. So we don't need for outliers treatment.

Features Scaling / Standard Scaler:

Feature Scaling is **a technique to standardize the independent features present in the data in a fixed range**. It is performed during the data pre-processing to handle highly varying magnitudes or values or units.

```
# Performing Standard scaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
X
```

```
array([[-1.67527846, -0.52966998,  0.17577689, ...,  0.74026909,
         0.67237607, -1.11171284],
       [-1.25535882,  0.72876416,  0.54772628, ...,  0.74026909,
        -1.26827214, -0.85948336],
       [-0.41551954, -0.68455418,  0.7867683 , ...,  0.74026909,
         2.18176913,  0.51631381],
       ...,
       [ 1.26415903, -1.11532587,  1.21991245, ...,  0.74026909,
         1.96614155, -0.05693501],
       [ 0.00440011, -0.18118053,  1.32987178, ...,  0.74026909,
         0.45674849,  0.97491287],
       [ 0.84423939, -0.55871077, -0.15792577, ...,  0.74026909,
         0.45674849, -1.20343265]])
```

By using a standard scaler, I have scaled the data in one range.

Building Machine Learning Models:

First, I will find the best random state on which I will get the maximum score.

```
maxScore = 0
maxRS = 0

for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=i)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    pred_train = lr.predict(x_train)
    pred_test = lr.predict(x_test)
    acc=r2_score(y_test,pred_test)
    if acc>maxScore:
        maxScore=acc
        maxRS=i
print('Best score is',maxScore,'on Random State',maxRS)
```

```
Best score is 0.5114352057819836 on Random State 65
```

Applying train-test split with Best Random State and applying ML on Different Algorithms:

```
model = [LinearRegression(),Lasso(alpha=1.0),Ridge(alpha=1.0),DecisionTreeRegressor(criterion='squared_error'),
         KNeighborsRegressor()]
for i in model:
    X_train1,X_test1,y_train1,y_test1 = train_test_split(X,y, test_size = 0.25, random_state =maxRS)
    i.fit(X_train1,y_train1)
    pred = i.predict(X_test1)
    print('Train Score of', i , 'is:' , i.score(X_train1,y_train1))
    print("r2_score", r2_score(y_test1, pred))
    print("mean_squred_error", mean_squared_error(y_test1, pred))
    print("RMSE", np.sqrt(mean_squared_error(y_test1, pred)),"\n")
```

```
Train Score of LinearRegression() is: 0.4457805727969435
r2_score 0.5114352057819836
mean_squred_error 362295.46902039496
RMSE 601.9098512405284

Train Score of Lasso() is: 0.4457639554424073
r2_score 0.5111749877628557
mean_squred_error 362488.4338234328
RMSE 602.0701236761652

Train Score of Ridge() is: 0.44578052283247993
r2_score 0.5114169961822433
mean_squred_error 362308.9723582463
RMSE 601.9210682126405

Train Score of DecisionTreeRegressor() is: 1.0
r2_score 0.9133786486041886
mean_squred_error 64234.10671936759
RMSE 253.44448449190523

Train Score of KNeighborsRegressor() is: 0.8265299936287829
r2_score 0.7436734225357128
mean_squred_error 190079.102513834
RMSE 435.9806217182525
```

Conclusions:

Have checked Multiple Model and their score also. I have found that Decision tree regressor model is overfitting. Other models are working well. But KNeighborsRegressor is having less train and test score difference with least mean square error and least RMSE. Now i will check with the ensemble method to boost up score

Using Ensemble Technique to boost up score:

RandomForestRegressor:

```
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=100,random_state=maxRS,criterion='squared_error', min_samples_split=2, min_samples_lea
#RandomForestClassifier(100)---Default
rf.fit(X_train1,y_train1)
predrf=rf.predict(X_test1)
print('Train Score of', rf , 'is:' , rf.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predrf))
print("mean_squred_error", mean_squared_error(y_test1, predrf))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predrf)))
```

```
Train Score of RandomForestRegressor(random_state=65) is: 0.992142228456897
r2_score 0.9480528948680718
mean_squred_error 38521.401952727276
RMSE 196.26869835184436
```

AdaBoostRegressor:

```
from sklearn.ensemble import AdaBoostRegressor

ABr=AdaBoostRegressor( base_estimator=KNeighborsRegressor(),n_estimators=50,learning_rate=1.0,loss='linear',random_state=max
#RandomForestClassifier(50)---Default
ABr.fit(X_train1,y_train1)
predAbr=ABr.predict(X_test1)
print('Train Score of', ABr , 'is:' , ABr.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predAbr))
print("mean_squred_error", mean_squared_error(y_test1, predAbr))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predAbr)))
```

```
Train Score of AdaBoostRegressor(base_estimator=KNeighborsRegressor(), random_state=65) is: 0.91079138752551
r2_score 0.769914374515491
mean_squred_error 170620.11136758892
RMSE 413.06187353420665
```

GradientBoostingRegressor:

```
from sklearn.ensemble import GradientBoostingRegressor

Gradient_Boost=GradientBoostingRegressor(n_estimators=100,loss='squared_error',learning_rate=0.1,criterion='friedman_mse', m
#GradientBoostingRegressor(100)---Default
Gradient_Boost.fit(X_train1,y_train1)
predgb=Gradient_Boost.predict(X_test1)
print('Train Score of', Gradient_Boost , 'is:' , Gradient_Boost.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predgb))
print("mean_squred_error", mean_squared_error(y_test1, predgb))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predgb)),"\n")
```

```
Train Score of GradientBoostingRegressor() is: 0.8874196455796406
r2_score 0.890635077059324
mean_squred_error 81099.61368365414
RMSE 284.77993904707216
```

Conclusion:

Here we can see the least difference between train score and test score is coming in GradientBoostingRegressor.So model is working well with both train model and test model.

For AdaBoostRegressor, the difference is high, So the model is overfitting.

For RandomForestRegressor, the difference is also more as compared to GradientBoostingRegressor.

So selecting GradientBoostingRegressor as final model

Hyper Parameter Tuning:

Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance. It works by running multiple trials in a single training process.

We are using Randomsearchcv method for hyperparameter tuning to find best parameters for GradientBoostingRegressor.

```
#Using GradientBoostingRegressor for hyper parameter tuning
Gradient_Boost = GradientBoostingRegressor()
Para = {"n_estimators":[100,200,300,400],
        "learning_rate":[0.1,0.3,0.5],
        "max_depth" :[3,5,7,9,10]}

Rand_search = RandomizedSearchCV(Gradient_Boost,Para,cv = 6,scoring = "r2",n_jobs =-1,verbose = 2)
Rand_search.fit(X_train1,y_train1)
print(Rand_search.best_params_)
```

```
Fitting 6 folds for each of 10 candidates, totalling 60 fits
{'n_estimators': 400, 'max_depth': 5, 'learning_rate': 0.1}
```

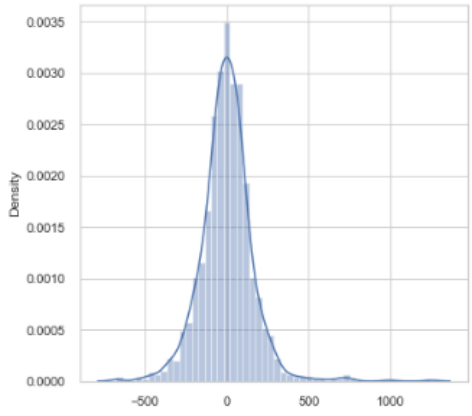
```
prediction = Rand_search.predict(X_test1)
```

```
Selling_Price = GradientBoostingRegressor(n_estimators= 400, max_depth= 5, learning_rate =0.1)
Selling_Price.fit(x_train, y_train)
pred = Selling_Price.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print("RMSE value:",np.sqrt(mean_squared_error(y_test, pred)))
```

```
R2_Score: 96.52564721843089
RMSE value: 154.02944685730992
```

The predicted y value is having a normalized curve which is good.

```
plt.figure(figsize = (6,6))
sns.distplot(y_test1-prediction)
plt.show()
```



As we can see that most of the residuals are 0, which means our model is generalizing well.

Cross Validation:

Cross-validation is **a resampling method that uses different portions of the data to test and train a model on different iterations**. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

```
best_Gradient_Boost = GradientBoostingRegressor(n_estimators= 400, max_depth= 5, learning_rate =0.1)

for i in range(2,11):
    cross_score = cross_val_score(best_Gradient_Boost,X,y,cv = i,n_jobs = -1)
    print(i,"mean",cross_score.mean() ,"and STD" , cross_score.std())
```

```
2 mean 0.837907362057158 and STD 0.0027278780051092633
3 mean 0.9221170173995855 and STD 0.02685668813429838
4 mean 0.9284414753401908 and STD 0.03911204829957108
5 mean 0.9385125254718284 and STD 0.04225479783247203
6 mean 0.9469831137799503 and STD 0.04100712570982601
7 mean 0.9407102030673815 and STD 0.03980250225281623
8 mean 0.9476470703338274 and STD 0.04044537706039697
9 mean 0.9512029166911673 and STD 0.03808853477585622
10 mean 0.9474124136165762 and STD 0.041539243692988365
```

Applying Cross validation Score=6

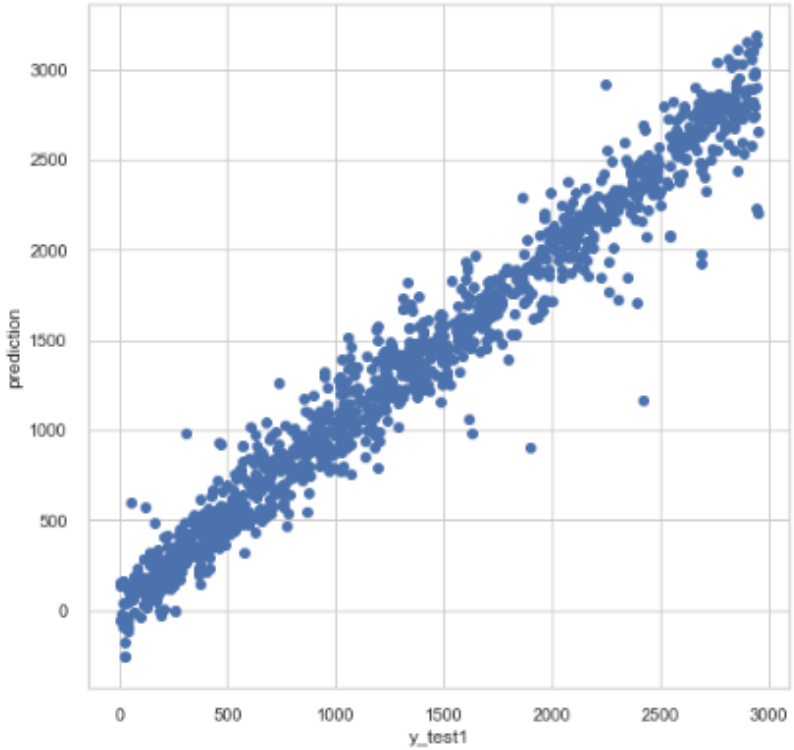
```
# Cross validate of RandomForestRegressor using cv=6
from sklearn.model_selection import cross_val_score
score=cross_val_score(best_Gradient_Boost,X,y,cv=6,scoring='r2')
print('Score:', score)
print('Mean Score:', score.mean())
print('Standard Deviation:', score.std())
```

```
Score: [0.99109706 0.98939463 0.96711154 0.93508075 0.92485076 0.87408461]
Mean Score: 0.94693655593532
Standard Deviation: 0.04101954186814009
```

Plotting y_test1 vs predictions:

- Simply plotting our predictions vs the true values.
- Ideally, it should be a straight line.

```
plt.figure(figsize = (8,8))
plt.scatter(y_test1, prediction)
plt.xlabel("y_test1")
plt.ylabel("prediction")
plt.show()
```



Saving the Model:

We are saving the model by using python's pickle library. It will be used further for the predictio n.

```
import pickle
# Saving the AdaBoostRegressor
best_Gradient_Boost.fit(X,y)
pred = best_Gradient_Boost.predict(X_test1)

# Saving model

filename = "OldCars_Saleprice_Prediction.pkl"
```

Here we have predicted the target of test dataset and checked the shape of test dataset and target of test dataset to join them.

CONCLUSION:

- After Scraping old car prices for cities like Delhi, Noida, Gurgaon, Mumbai, Pune, Hyderabad, Bangalore, Ahmedabad, Chennai, Kolkata from different websites like

Cars24 and CArdekho.com I have prepared an excel sheet and loaded the dataset for further EDA process.

- So, as we saw that we have done a complete EDA process, getting data insights, feature engineering, and data visualization as well so after all these steps one can go for the prediction using machine learning model-making steps.
- We have both numerical and categorical data types features in the datasets and the dependent variable of train data i.e. the **Selling price** is the numerical data type. So, I applied the regression method for prediction.
- Once data has been cleaned and missing value is replaced, Label encoding is applied to them to convert them into Numerical ones. I trained the model on five different algorithms but for most of the models, train and test data was having a variance, and the model was overfitting.
- Only Gradient Boost regressor worked well out of all the models, as there was less difference between train score and test score and RMSE was also low hence I used it as the final model and have done further processing.
- After applying hyperparameter tuning I got an accuracy(r2_score) of 96% from the GradientBoostRegressor model after hyper parameter tuning which is a good score.

Then I saved the model.

Limitations and Scope:

- This study used different models in order to predict used car prices. However, there was a relatively small dataset for making a strong inference because number of observations was only 5058. Gathering more data can yield more robust predictions.
- Secondly, there could be more features that can be good predictors. For example, here are some variables that might improve the model: number of doors, gas/mile (per gallon), color, mechanical and cosmetic reconditioning time, used-to-new ratio, appraisal-to-trade ratio.
- Another point that has room to improve is that the data cleaning process can be done more rigorously with the help of more technical information. For example, instead of using 'fill' method, there might be indicators that helps to fill missing values more meaningfully.

As a suggestion for further studies, while pre-processing data, instead of using a label encoder, one hot encoder method can be used. Thus, all non-numeric features can be converted to nominal data instead of ordinal data (Raschka & Mirjalili, 2017).

I hope this article helped you to understand Data Analysis, Data Preparation, and Model building approaches in a much simpler way.

Thank you