Car Price Prediction

Submitted by:

Dipesh Ramesh Limaje

Problem Statement

With the covid 19 impact on 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 our clients works with small traders, who sell used cars. With the change in the 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 a car price valuation model.

Conceptual Background of the Domain Problem

The domain problem of car price prediction in data science involves predicting the prices of cars based on various features, such as make and model, year of manufacture, number of kilometers driven, brand, transmission type, and ownership history, among others. The objective of the model is to provide accurate predictions of car prices that can be used by individuals, dealerships, and manufacturers to make informed decisions.

In order to solve this problem, a wide range of statistical and machine learning techniques can be used, such as linear regression, decision trees, random forests, gradient boosting, and neural networks, among others. The choice of technique depends on the complexity of the problem, the size of the data, and the computational resources available. The models are trained on historical data, and the accuracy of the model is evaluated based on how well it predicts the prices of cars in the test data set.

The success of the car price prediction model depends on the availability of high-quality data and the ability to extract meaningful features from it. Feature engineering and data pre-processing techniques, such as normalization, scaling, and imputation, are crucial steps in the development of a high-performing model. Additionally, it is important to address any issues with the data, such as missing values, outliers, and skewness, in order to ensure that the model provides accurate predictions.

Review of Literature

In the field of data science, car price prediction has been a topic of interest for researchers and practitioners for many years. Over the years, numerous studies have been conducted on the development and evaluation of car price prediction models.

Many studies have used traditional statistical techniques, such as linear regression, to build car price prediction models. These models have been found to perform well in certain situations, but their limitations become apparent when dealing with complex data sets with multiple variables.

In recent years, machine learning techniques, such as decision trees, random forests, and gradient boosting, have gained popularity for car price prediction. These models have been found to perform well in complex scenarios and to be more flexible than traditional statistical models. Additionally, they are capable of handling large amounts of data and can be used to uncover complex relationships between variables.

Some studies have explored the use of neural networks for car price prediction. These models have been found to perform well in complex scenarios and to be capable of capturing complex relationships between variables. However, they are computationally intensive and require a large amount of data to train effectively.

In conclusion, the literature review suggests that there is a wide range of techniques that can be used for car price prediction in data science, with each technique having its own advantages and limitations. It is important to select the most appropriate technique for each particular problem and to carefully consider the quality and complexity of the data when building a car price prediction model.

Motivation for the Problem Undertaken

The motivation for the problem of car price prediction in data science is driven by several factors, including the need to accurately estimate the value of a car, the importance of understanding the factors that influence car prices, and the increasing demand for more sophisticated methods for valuing vehicles.

One of the main motivations for this problem is the need to accurately estimate the value of a car. The price of a car can have a significant impact on its owner, both financially and emotionally. Accurately predicting the price of a car can help its owner make informed decisions about whether to buy or sell, and can also help car dealers make more informed decisions about the vehicles they purchase and sell.

Another motivation for this problem is the importance of understanding the factors that influence car prices. There are many factors that can impact the price of a car, including its make and model, its age, its condition, and its location. By understanding the relationship between these factors and car prices, car owners, car dealers, and other stakeholders can make more informed decisions about buying and selling vehicles.

Finally, there is a growing demand for more sophisticated methods for valuing vehicles. As the market for used cars continues to grow, the need for more accurate and reliable methods for valuing vehicles becomes increasingly important. The development of car price prediction models in data science can help meet this demand by providing a more sophisticated and accurate means of estimating the value of a vehicle.

Mathematical/ Analytical Modelling of the Problem

The mathematical and analytical modeling of the problem of car price prediction in data science is typically based on regression analysis. This involves using a set of input variables (also known as features or predictors) to predict the value of a target variable, which in this case is the price of a car.

The first step in creating a car price prediction model is to gather data on the variables that are believed to impact car prices. This data is then used to build a model that can accurately predict the price of a car based on its features.

One common approach is to use multiple linear regression, which involves using a linear equation to model the relationship between the input variables and the target variable. In this type of model, the target variable is modelled as a linear combination of the input variables, with each input variable being weighted by a coefficient that represents its importance in determining the value of the target variable.

Another common approach is to use non-linear regression, which involves using a non-linear equation to model the relationship between the input variables and the target variable. This type of model is more flexible than multiple linear regression and can better capture complex relationships between the input variables and the target variable.

In addition to regression analysis, other modelling techniques that can be used for car price prediction include decision trees, random forests, support vector machines, and neural networks. The choice of modelling technique will depend on the nature of the data, the complexity of the relationships between the variables, and the desired level of accuracy.

Data Sources and their formats

The data sources for car price prediction in data science can be obtained through web scraping. Web scraping is a process of automatically extracting data from websites. In this case, data can be scraped from websites that provide information on cars such as manufacturer websites, car dealership websites, and automotive review websites. The data obtained can be in different formats such as CSV, JSON, or HTML. The data should include features such as make and model of the car, year of manufacture, engine type, transmission type, fuel type, mileage, and the asking price.

Data Pre-processing Done

Pre-processing

```
In [238]: #checking the datatypes
          df.dtypes

Out[238]: Brand                int32
          Model                int32
          Manu_Year            int64
          Driven_kilometer     int64
          Fuel                 int32
          Transmission         int32
          Number_of_owners     int64
          Location             int32
          Price              float64
          dtype: object
```

```
In [24]: #Dropping unwanted column
         df.drop('Unnamed: 0' , axis=1,inplace=True)
```

```
In [25]: #checking the shape of dataset
         df.shape

Out[25]: (5463, 9)
```

```
In [27]: #checking nulls present in dataset
         df.isnull().sum()

Out[27]: Brand                0
         Model                0
         Manu_Year            0
         Driven_kilometer     0
         Fuel                 0
         Transmission       479
         Number_of_owners     0
         Location             0
         Price              781
         dtype: int64
```

Pre-processing I have done

1] Check for the Null values
2] Check the Shape of the dataset
3] Check all the column's Names and Compared them with the Datatypes.

4] Then I have check for the duplicates and remove the duplicate by drop duplicate method

Checking the duplicates

In [28]:
```
#  check  the duplicate
duplicate = df[df.duplicated()]
print("Duplicate Rows :")

#  Print the resultant Dataframe
duplicate
```

```
Duplicate Rows :
```

Out[28]:

	Brand	Model	Manu_Year	Driven_kilometer	Fuel	Transmission	Number_of_owners	Location	Price
20	Maruti	Ertiga	2016	85970	Diesel	NaN	1	Bangalore	817000.0
21	Tata	Tiago	2018	79628	Petrol	Automatic	1	Bangalore	483000.0
22	Volkswagen	Vento	2018	43521	Petrol	Automatic	1	Bangalore	1094000.0
23	Tata	Tiago	2018	28154	Petrol	Manual	1	Bangalore	624000.0
24	Renault	Kwid	2019	13588	Petrol	Automatic	1	Bangalore	543000.0
...	...	...	...	...	...	...	...	...	...
4498	Hyundai	New Elantra	2017	36474	Petrol	Manual	1	Mumbai	NaN
4499	Hyundai	Grand i10	2014	42171	Petrol	Automatic	1	Mumbai	436000.0
4500	Hyundai	NEW SANTRO	2019	46122	Petrol	NaN	1	Mumbai	514000.0
4501	Volkswagen	Polo	2018	32263	Petrol	Manual	1	Mumbai	592000.0
4502	Mercedes	Benz E Class	2011	78206	Diesel	Automatic	2	Mumbai	NaN

240 rows × 9 columns

In [29]:
```
# dropping the duplicates
df.drop_duplicates(inplace=True)
```

In [30]:
```
#  check  the duplicate again
duplicate = df[df.duplicated()]
print("Duplicate Rows :")

#  Print the resultant Dataframe
duplicate
```

```
Duplicate Rows :
```

Out[30]:

Brand	Model	Manu_Year	Driven_kilometer	Fuel	Transmission	Number_of_owners	Location	Price

5] Then I have observe Null values in target Variable i.e Price of car so I don't know the exact value of that particular car so rather to fill the values with mean method lets drop the null values present in the target variable.

Filling Null values

In [33]:
```
df['Transmission']=df['Transmission'].fillna(df['Transmission'].mode()[0])
```

In [34]:
```
#checking nulls present in dataset
df.isnull().sum()
```

Out[34]:
```
Brand                0
Model                0
Manu_Year            0
Driven_kilometer     0
Fuel                 0
Transmission         0
Number_of_owners     0
Location             0
Price              757
dtype: int64
```

So in our label column we have 757 null values, so its our target varaible and we dont know exact value of that 757 cars so rather filling the values by mean we directly dropping that 757 rows and keep our dataset safe.

In [ ]:
```

```

Dropping rest of the null values

In [37]:
```
df.dropna(inplace=True)
```

In [38]:
```
df.shape
```

Out[38]:
```
(4466, 9)
```

6] After that I have to Describe the dataset to observe the numerical values and write the Observations.

dtype: object

In [82]:
```
#describe dataset
df.describe()
```

Out[82]:

	Brand	Model	Manu_Year	Driven_kilometer	Fuel	Transmission	Number_of_owners	Location	Price
count	4466.000000	4466.000000	4466.000000	4466.000000	4466.000000	4466.000000	4466.000000	4466.000000	4.466000e+03
mean	8.832288	113.182490	2017.329825	42269.418271	2.505598	1.770712	1.240484	5.750336	5.920711e+05
std	3.811969	43.098196	2.762457	26391.484623	1.027980	0.636552	0.445890	3.936467	2.707564e+05
min	0.000000	0.000000	2009.000000	215.000000	0.000000	0.000000	1.000000	0.000000	1.470000e+05
25%	5.000000	87.000000	2015.000000	22085.000000	3.000000	2.000000	1.000000	2.000000	4.200000e+05
50%	11.000000	112.000000	2018.000000	37772.000000	3.000000	2.000000	1.000000	5.000000	5.400000e+05
75%	11.000000	148.000000	2019.000000	60132.750000	3.000000	2.000000	1.000000	10.000000	6.867500e+05
max	17.000000	185.000000	2022.000000	680010.000000	3.000000	2.000000	3.000000	12.000000	2.866000e+06

Observations

1] There are 5463 rows and 10 columns and after treating we got 4466 row and 9 columns

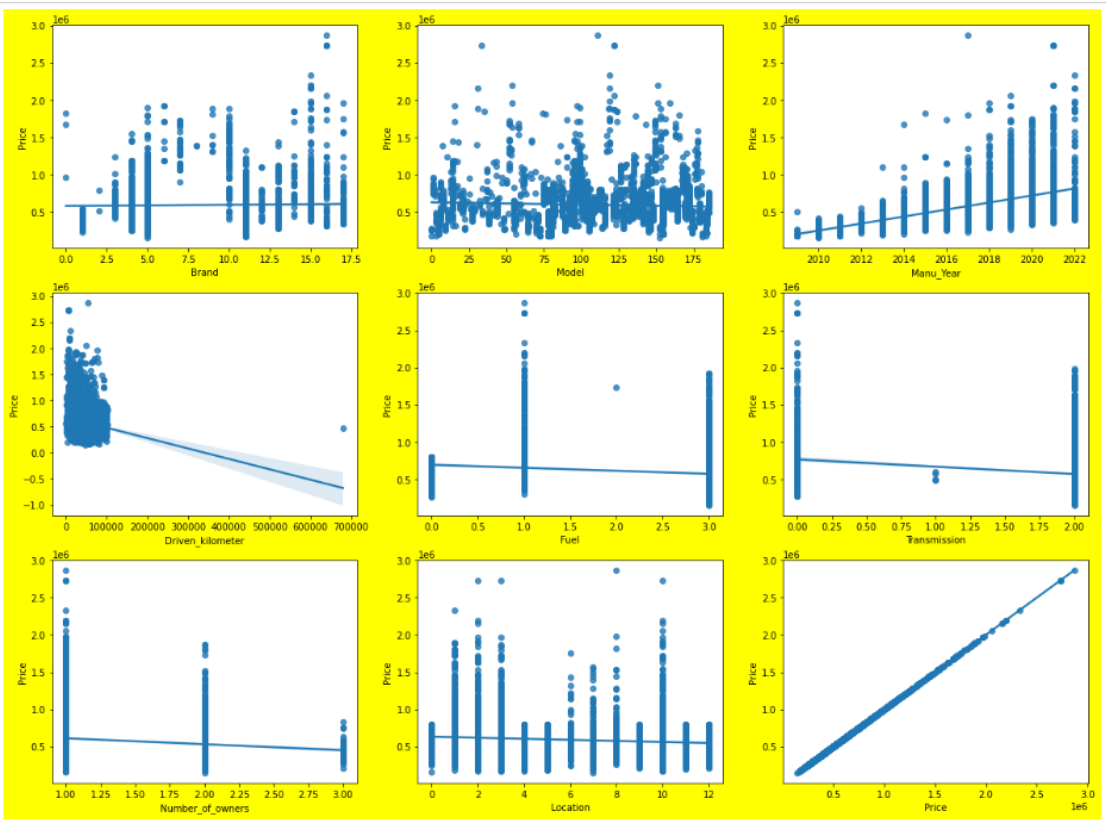
2] There was Null values in Transmission columns so we treat it with mode method

3] There was null values in our target varable so we drop that null beacuse we cant directly judge car price by mean method

4] There were 240 duplicates in our dataset so we drop the duplicates

5] We observe problem in Driven_kilometer column beacuse Standrad deviation is somewhat 50% of mean so our data is scattered and min value is 215km which can be possible but max value is 680010km which i guess cannot possible

Data Inputs- Logic- Output Relationships



To observe the relationship between Feature and label so I created this Regression plot to observe which features are positively co-related and which features are negatively co-related.

State the set of assumptions (if any) related to the problem under consideration

1] For this particular problem I have dropped the duplicate value which I have found.
2] For this particular problem I have assumed that the Maximum VIF should be 5, if any of the features has a VIF which is greater than 5 we should drop that feature.
3] I have directly drop the null values that are present in our target variable.

Hardware and Software Requirements and Tools Used

Hardware Requirements: -Computer with minimum 8 GB RAM -High-speed internet connection -High-end graphics card -External storage device

Software Requirements: -Python programming language -TensorFlow -Keras -Scikit-Learn -Pandas -Matplotlib -Seaborn

Tools Used: -Jupyter Notebook -Google Collab -Tableau -Power BI

Predicting the price of a Used Cars : -Linear regression -Random Forest -GBoost -ADA Boost.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

There are several problem-solving approaches that can be used in the context of car price prediction in data science:

- Linear Regression: A simple and straightforward method for modeling the relationship between a dependent variable (car price) and one or multiple independent variables (features such as make, model, year, etc.).
- Decision Tree Regression: This method builds a tree-like model that considers the features and their interactions to predict the target (car price).
- Random Forest Regression: This method is an extension of decision tree regression and combines the predictions of multiple decision trees to improve the overall accuracy of the model.
- Gradient Boosting Regression: This method trains weak models sequentially and combines them to form a stronger model.
- Neural Networks: This method uses artificial neural networks to model the relationship between inputs and outputs.
- Support Vector Regression: This method uses support vector machines to find the optimal line (or hyperplane) that best separates the data into two classes.
- K-Nearest Neighbors Regression: This method uses the k-nearest neighbors of a given data point to predict its target value (car price).

These are some of the common methods used for car price prediction in data science, and the best approach will depend on the specific problem, data, and objectives.

Testing of Identified Approaches (Algorithms)

- ➢ LR (Linear Regression Model)
- ➢ GBDT (Gradient Boosting Regressor Model)
- ➢ RF (Random Forest Regressor Model)
- ➢ ADA (AdaBoost Regressor Model)

Run and Evaluate selected models

1st Model I have Created is the Logistic Regression Model

LinearRegression Model

In [125]:
```
#Lets import necessary library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

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

Best Random State

In [126]:
```
#Best Random State
MaxAccu=0
MaxRS=0

for i in range (0,200):
    X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=i)
    regression=LinearRegression()
    regression.fit(X_train,y_train)

    pred=regression.predict(X_train)
    training=regression.score(X_train,y_train)
    print ('Training Score' , training*100 , 'RandomState' ,i)

    y_pred=regression.predict(X_test)
    testing=regression.score(X_test,y_test)
    print ('Testing Score' , testing*100 , 'RandomState' ,i)
    print('\n')


    if testing>MaxAccu:
        MaxAccu=testing
        MaxRS=i
        print('MAXINING TESTING SCORE' , MaxAccu*100 , 'ON RANDOM STATE OF' , i)
```

```
Training Score 30.112780657039128 RandomState 0
Testing Score 29.616048275962914 RandomState 0


MAXINING TESTING SCORE 29.616048275962914 ON RANDOM STATE OF 0
Training Score 30.04204624484744 RandomState 1
Testing Score 29.662511184652207 RandomState 1


MAXINING TESTING SCORE 29.662511184652207 ON RANDOM STATE OF 1
Training Score 29.538395490548343 RandomState 2
Testing Score 31.118595205594335 RandomState 2
```

Training the Model

In [128]:
```
#splliting our data into train test split and randomstate 8
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=49)
```

In [129]:
```
#Training the data on Linear Regression Model
regression=LinearRegression()
regression.fit(X_train,y_train)
```

Out[129]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [130]:
```
#training score
regression.score(X_train,y_train)
```

Out[130]:
```
0.28461352497162873
```

In [131]:
```
#testing score
regression.score(X_test,y_test)
```

Out[131]:
```
0.3446139759865413
```

Model Score

```
Training Score = 28.461352497162873 %
Testing Score = 34.46139759865413 %
```

In [ ]:

LR (Linear Regression Model) Score are

Training Score = 28.461352497162873 %
Testing Score = 34.46139759865413 %

LASSO MODEL

In [136]:
```
#import library
from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV
```

In [137]:
```
##### LASSO MODEL######

lasscv = LassoCV(alphas = None , max_iter = 100)

lasscv.fit(X_train , y_train)
```

Out[137]:
```
LassoCV(max_iter=100)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [138]:
```
# best aplha parameter
alpha = lasscv.alpha_
alpha
```

Out[138]:
```
360.33811300106515
```

In [139]:
```
# now we have best parametr noe train according to it
lasso_reg = Lasso(alpha)
lasso_reg.fit(X_train,y_train)
```

Out[139]:
```
Lasso(alpha=360.33811300106515)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [140]:
```
# now check r2 score
lasso_reg.score(X_test,y_test)
```

Out[140]:
```
0.34433061007683285
```

RIDGE MODEL

In [141]:
```
############ RIDGE MODEL#########

ridgecv = RidgeCV(alphas = np.arange(0.001,0.1,0.01))
ridgecv.fit(X_train , y_train)
```

Out[141]:
```
RidgeCV(alphas=array([0.001, 0.011, 0.021, 0.031, 0.041, 0.051, 0.061, 0.071, 0.081,
       0.091]))
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [142]:
```
# best aplha parameter
alpha = ridgecv.alpha_
alpha
```

Out[142]:
```
0.09099999999999998
```

In [143]:
```
# now we have best parametr noe train according to it
ridge_reg = Ridge(alpha)
ridge_reg.fit (X_train,y_train)
```

Out[143]:
```
Ridge(alpha=0.09099999999999998)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [144]:
```
# now check r2 score
ridge_reg.score(X_test,y_test)
```

Out[144]:
```
0.34461187665015414
```

LASSO SCORES = 34.433061007683285 %
RIDGE SCORES = 34.461187665015414 %

2nd Model I have Created is Ada Boost Regressor Model

AdaBoostRegressor Model

In [145]:
```
# IMPORT LIBRARY
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostRegressor
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
%matplotlib inline

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

Best Random State

In [146]:
```
#Best Random State
MaxAccu=0
MaxRS=0

for i in range (0,200):
    X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=i)
    ada=AdaBoostRegressor()
    ada.fit(X_train,y_train)

    pred=ada.predict(X_train)
    training=ada.score(X_train,y_train)
    print ('Training Score' , training*100 , 'RandomState' ,i)

    y_pred=ada.predict(X_test)
    testing=ada.score(X_test,y_test)
    print ('Testing Score' , testing*100 , 'RandomState' ,i)
    print('\n')


    if testing>MaxAccu:
        MaxAccu=testing
        MaxRS=i
        print('MAXINING TESTING SCORE' , MaxAccu*100 , 'ON RANDOM STATE OF' , i)
```

```
Training Score 32.077371482492076 RandomState 0
Testing Score 29.032289272669665 RandomState 0


MAXINING TESTING SCORE 29.032289272669665 ON RANDOM STATE OF 0
Training Score 29.494347802977 79 RandomState 1
```

Training the model

In [154]:
```
#splliting our data into train test split and randomstate 8
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=126)
```

In [155]:
```
# adaboost inilize
from sklearn.ensemble import AdaBoostRegressor
ada=AdaBoostRegressor()
ada.fit(X_train,y_train)
```

Out[155]:
```
AdaBoostRegressor()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [156]:
```
 # model prediction on training dataset
y_pred = ada.predict(X_train)
```

In [157]:
```
accuracy = metrics.r2_score (y_train , y_pred)
print ('R Squared Score : ' , accuracy)
```

```
R Squared Score :  0.37670083153570255
```

In [158]:
```
# model prediction on testing datadet
pred = ada.predict(X_test)
```

In [159]:
```
accuracy = metrics.r2_score(y_test,pred)
print ('R Squared Score : ' , accuracy)
```

```
R Squared Score :  0.404813989022475
```

In [ ]:

Ada Boost Regressor Model Scores

Training Score = 37.670083153570255 %

Testing Score = 40.4813989022475 %

Hyperparameter Tuning for Ada Boost

In [160]:
```
### HYPERPARAMETER TUNING ###
from sklearn.model_selection import RandomizedSearchCV
```

In [161]:
```
params = {'n_estimators': [45,47,53,55,60,70] ,
          'learning_rate':[0.25,0.30,0.40]}
```

In [162]:
```
rnd_srch = RandomizedSearchCV(AdaBoostRegressor() , cv=5 , param_distributions=params , n_jobs=-1)
```

In [163]:
```
rnd_srch.fit(X_train,y_train)
```

Out[163]:
```
RandomizedSearchCV(cv=5, estimator=AdaBoostRegressor(), n_jobs=-1,
                   param_distributions={'learning_rate': [0.25, 0.3, 0.4],
                                        'n_estimators': [45, 47, 53, 55, 60,
                                                         70]})
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [164]:
```
rnd_srch.best_params_
```

Out[164]:
```
{'n_estimators': 45, 'learning_rate': 0.25}
```

In [165]:
```
rnd_srch.best_estimator_
```

Out[165]:
```
AdaBoostRegressor(learning_rate=0.25, n_estimators=45)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [167]:
```
ada = AdaBoostRegressor(learning_rate=0.41, n_estimators=62)
ada.fit(X_train,y_train)

pred=ada.predict(X_train)
print('====Training Score====')
print(metrics.r2_score(y_train,pred))
y_pred = ada.predict(X_test)

print ('=== Testing Score ===')
print (metrics.r2_score(y_test,y_pred))
```

```
====Training Score====
0.4502410016984354
=== Testing Score ===
0.4529517945982948
```

In [ ]:

\

Model Score after Hyperparameter Tuning

Training Score = 45.02410016984354 %

Testing Score = 45.29517945982948 %

3rd Model I have Created is Random Forest Regressor

RandomForestRegressor Model

In [168]:
```
#import necessary library

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import seaborn as sns

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

Best Random State

In [169]:
```
#Best Random State
MaxAccu=0
MaxRS=0

for i in range (0,200):
    X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=i)
    rf=RandomForestRegressor()
    rf.fit(X_train,y_train)

    pred=rf.predict(X_train)
    training=rf.score(X_train,y_train)
    print ('Training Score' , training*100 , 'RandomState' ,i)

    y_pred=rf.predict(X_test)
    testing=rf.score(X_test,y_test)
    print ('Testing Score' , testing*100 , 'RandomState' ,i)
    print('\n')


    if testing>MaxAccu:
        MaxAccu=testing
        MaxRS=i
        print('MAXINING TESTING SCORE' , MaxAccu*100 , 'ON RANDOM STATE OF' , i)
```

```
Training Score 98.02386706456097 RandomState 0
Testing Score 82.13204421666734 RandomState 0


MAXINING TESTING SCORE 82.13204421666734 ON RANDOM STATE OF 0
Training Score 97.71199465399714 RandomState 1
Testing Score 85.07490951793162 RandomState 1
```

Training the model

In [171]:
```
#splliting our data into train test split and randomstate 8
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=45)
```

In [172]:
```
rf=RandomForestRegressor()
rf.fit(X_train,y_train)
```

Out[172]:
```
RandomForestRegressor()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [173]:
```
 # model prediction on training dataset
y_pred = rf.predict(X_train)
```

In [174]:
```
accuracy = metrics.r2_score (y_train , y_pred)
print ('R Squared Score : ' , accuracy)
```

```
R Squared Score :  0.9761853281129097
```

In [175]:
```
# model prediction on testing datadet
pred = rf.predict(X_test)
```

In [176]:
```
accuracy = metrics.r2_score(y_test,pred)
print ('R Squared Score : ' , accuracy)
```

```
R Squared Score :  0.9031153190774422
```

Model Score

```
Training Score = 97.61853281129097 %
Testing Score = 90.31153190774422 %
```

Random Forest Regressor Model Score

Training Score = 97.61853281129097 %
Testing Score = 90.31153190774422 %

Hyperparameter tuning for Random Forest

In [177]:
```
# RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
```

In [178]:
```
# define parameters
parameters={'criterion':['mse','mae','poisson'],
            'max_features':['auto','sqrt','log2'],
            'min_samples_split':[1,11],
            'max_depth':[1,15],
            'min_samples_leaf':[1,7]}
```

In [179]:
```
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(X_train,y_train)
```

Out[179]:
```
GridSearchCV(estimator=RandomForestRegressor(),
             param_grid={'criterion': ['mse', 'mae', 'poisson'],
                         'max_depth': [1, 15],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 7],
                         'min_samples_split': [1, 11]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [180]:
```
#print best parameters
print(clf.best_params_)
```

```
{'criterion': 'poisson', 'max_depth': 15, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 1}
```

In [209]:
```
#reassign best parameters
rf=RandomForestRegressor(criterion= 'poisson', max_depth= 15, max_features= 'auto', min_samples_leaf= 1, min_samples_split= 1)
rf.fit(X_train,y_train)
```

Out[209]:
```
RandomForestRegressor(criterion='poisson', max_depth=15, max_features='auto',
                      min_samples_split=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [210]:
```
from sklearn.metrics import r2_score
print ('Training R2 Score: ' ,rf.score(X_train,y_train)*100)
```

```
Training R2 Score:  97.31488393187922
```

In [211]:
```
pred_decision=rf.predict(X_test)
rfs = r2_score(y_test,pred_decision)
```

In [212]:
```
print('Testing R2 Score:' , rfs*100)
```

```
Testing R2 Score: 89.82695266886329
```

Model Score after Hyperparameter Tuning

Training Score = 97.31488393187922 %

Testing Score = 89.82695266886329 %

4th Model I have Created is Gradient Boosting Regressor Model

GradientBoostingRegressor Model

In [213]:
```
# import library

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectPercentile , chi2
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
```

Best Random State

In [214]:
```
#Best Random State
MaxAccu=0
MaxRS=0

for i in range (0,200):
    X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=i)
    gbdt=GradientBoostingRegressor()
    gbdt.fit(X_train,y_train)

    pred=gbdt.predict(X_train)
    training=gbdt.score(X_train,y_train)
    print ('Training Score' , training*100 , 'RandomState' ,i)

    y_pred=gbdt.predict(X_test)
    testing=gbdt.score(X_test,y_test)
    print ('Testing Score' , testing*100 , 'RandomState' ,i)
    print('\n')


    if testing>MaxAccu:
        MaxAccu=testing
        MaxRS=i
        print('MAXINING TESTING SCORE' , MaxAccu*100 , 'ON RANDOM STATE OF' , i)
```

```
Training Score 82.63587131111925 RandomState 0
Testing Score 74.9816076553421 RandomState 0


MAXINING TESTING SCORE 74.9816076553421 ON RANDOM STATE OF 0
Training Score 82.13778296647989 RandomState 1
```

training the model

In [216]:
```
#splliting our data into train test split and randomstate 8
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.25,random_state=45)
```

In [217]:
```
# initiate GradientBoostingClassifier
gbdt= GradientBoostingRegressor()
gbdt.fit(X_train , y_train)
```

Out[217]:
```
GradientBoostingRegressor()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [218]:
```
 # model prediction on training dataset
y_pred = gbdt.predict(X_train)
```

In [219]:
```
from sklearn.metrics import r2_score
import sklearn.metrics as metrics
accuracy = metrics.r2_score (y_train , y_pred)
print ('R Squared Score : ' , accuracy)
```

```
R Squared Score :  0.8107134108184639
```

In [220]:
```
# model prediction on testing datadet
pred = gbdt.predict(X_test)
```

In [221]:
```
accuracy = metrics.r2_score(y_test,pred)
print ('R Squared Score : ' , accuracy)
```

```
R Squared Score :  0.8277657900200992
```

Model Score

```
Training Score = 81.07134108184639 %
Testing Score = 82.77657900200992 %
```

Gradient Boosting Regressor Model Score

Training Score = 81.07134108184639 %

Testing Score = 82.77657900200992 %

Hyperparameter tuning for GradientBoostingRegressor

In [ ]:
```
# HYPERPARAMETER TUNING #
from sklearn.model_selection import GridSearchCV
```

In [222]:
```
# internally it will use decision tree as name suggest GBDT and here we are going to add one new parameter i.e learning rate

grid_params = {'max_depth' : range(1,8),
               'min_samples_split': range(2,12,1),
               'learning_rate': np.arange(0.1 , 0.9),
               'n_estimators': [90,95,100,105,110]}
```

In [223]:
```
grid = GridSearchCV(GradientBoostingRegressor() , param_grid = grid_params , n_jobs = -1)
```

In [224]:
```
grid.fit(X_train,y_train)
```

Out[224]:
```
GridSearchCV(estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': array([0.1]),
                         'max_depth': range(1, 8),
                         'min_samples_split': range(2, 12),
                         'n_estimators': [90, 95, 100, 105, 110]})
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [225]:
```
grid.best_params_
```

Out[225]:
```
{'learning_rate': 0.1,
 'max_depth': 7,
 'min_samples_split': 5,
 'n_estimators': 105}
```

In [226]:
```
gbdt_clf = GradientBoostingRegressor(learning_rate= 0.1,
 max_depth= 7,
 min_samples_split= 5,
 n_estimators= 105)
```

In [227]:
```
gbdt_clf.fit(X_train,y_train)
```

Out[227]:
```
GradientBoostingRegressor(max_depth=7, min_samples_split=5, n_estimators=105)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [228]:
```
 # model prediction on training dataset
y_pred = gbdt_clf.predict(X_train)
```

In [229]:
```
accuracy = metrics.r2_score (y_train , y_pred)
print ('R Squared Score : ' , accuracy)
```

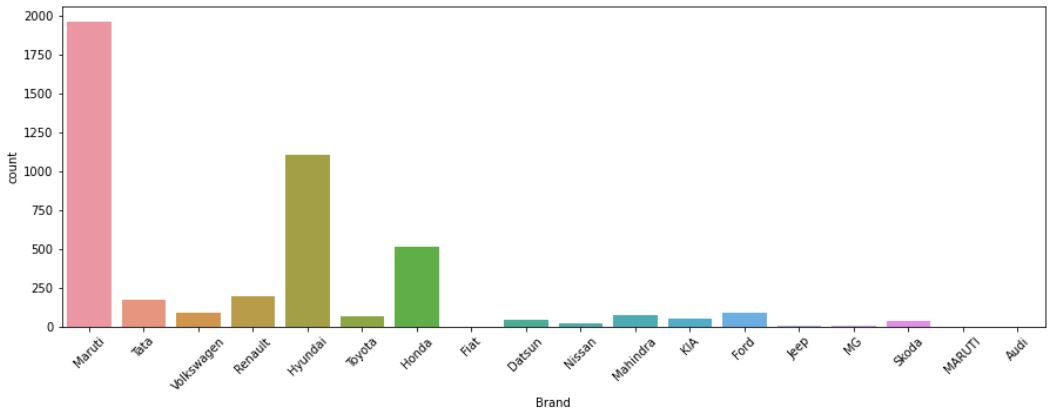
```
R Squared Score :  0.9804335166709452
```

Model Score after Hyperparameter Tuning

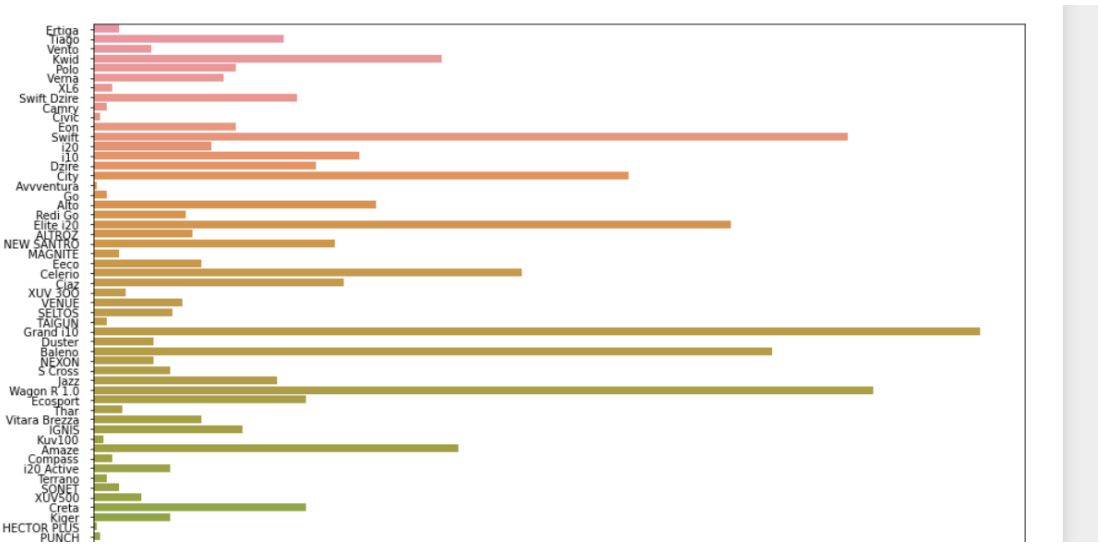
Training Score = 98.04335166709452 %

Testing Score = 90.42506463306986 %

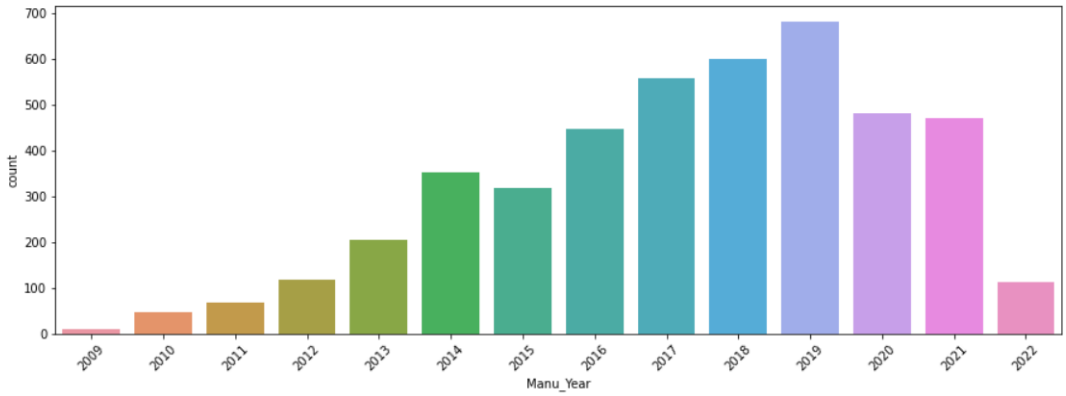
Visualizations and EDA



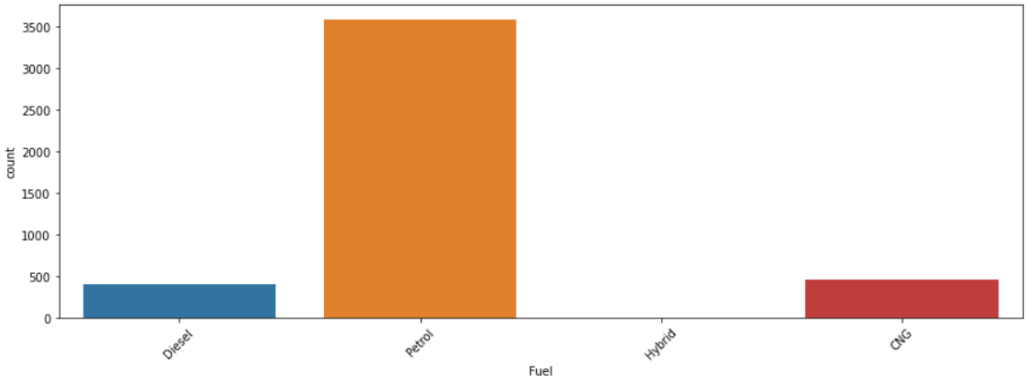
From the above bar graph, we observe there are 18 brands in our dataset and Maruti has the highest number of car in our dataset which is 1964 cars followed by Hyundai with 1106 cars and Honda with 517 cars.



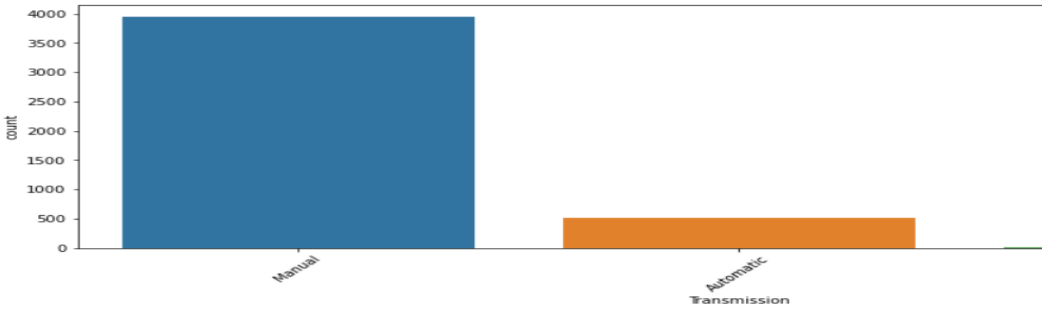
From the above bar graph, we observe the Most listed cars which are the Grand i10 with 280 cars followed by WagnoR with 246 cars, and followed by Swift with 238 cars listed on the Cars24 site.



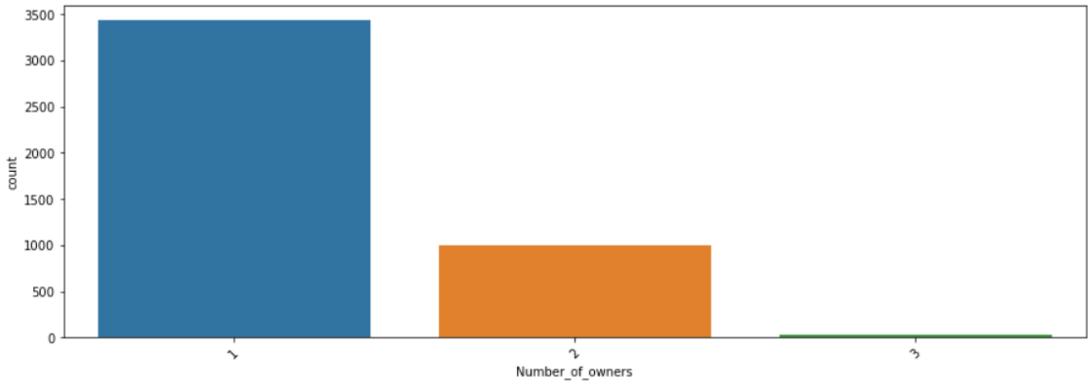
From the above bar graph, we observe the manufacturing year of the cars the most car listed in cars24 is of the 2019 year with 681 cars followed by the 2018 year with 599 cars, followed by 2017 with 556 cars.



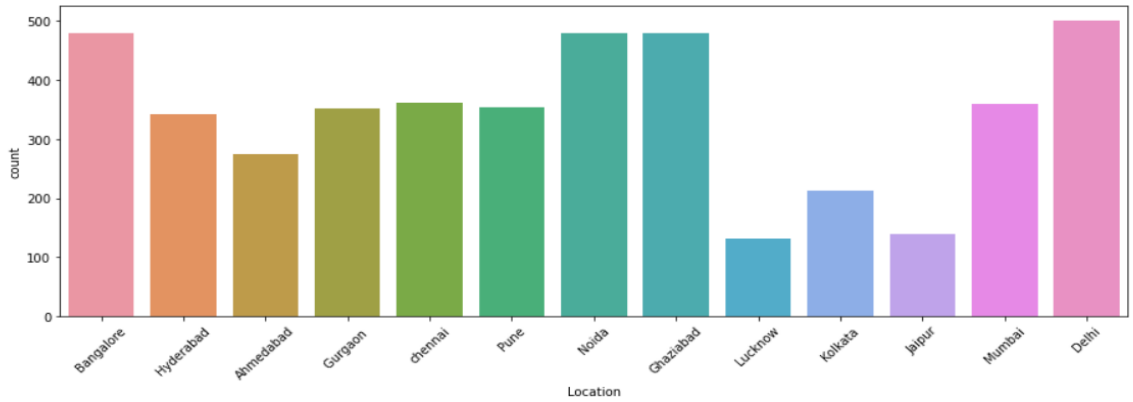
From the above bar graph we observe Fuel of the cars and cars listed in car24 is mostly having fuel type petrol with 3552 cars followed by Diesel cars with 464 cars, followed by CNG cars and Hybrid cars with 401 and 1 car.



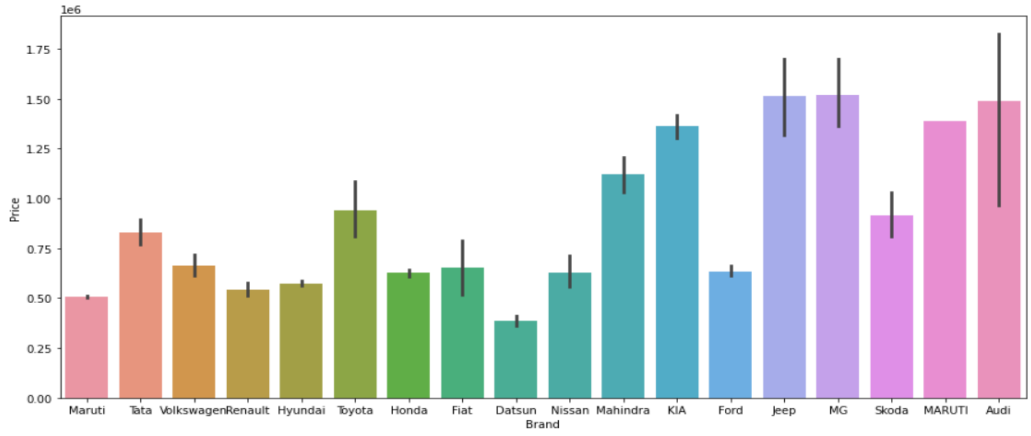
From the above bar graph we observe Manual Cars are listed more than automatic cars and count of the manual car is 3956 cars and the count of the automatic car is 510.



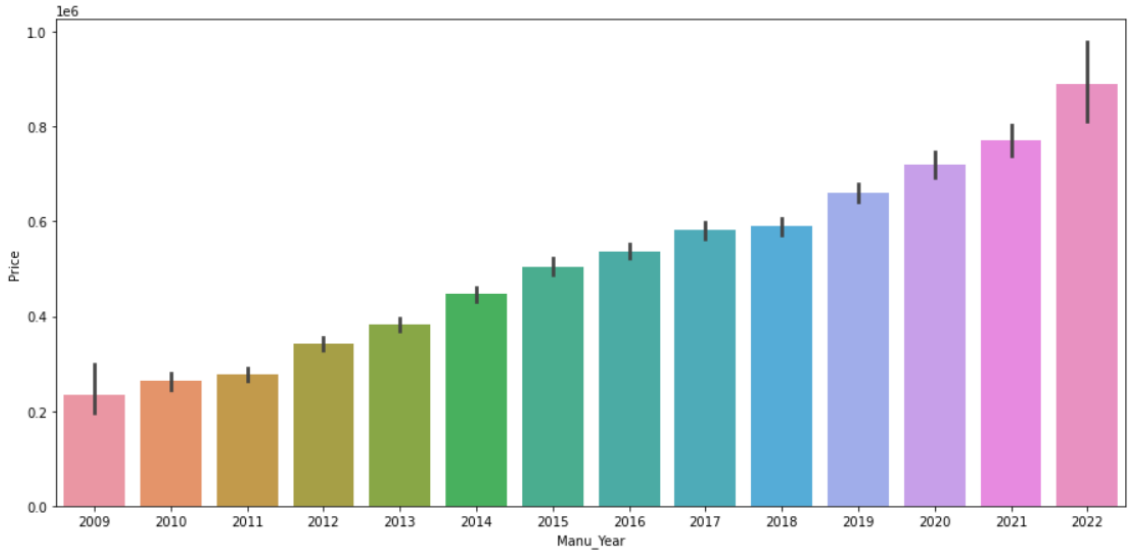
From the above bar graph, we observe 3428 cars is having 1st owners where 1002 cars have 2nd owners and 36 cars are having 3rd owners cars.



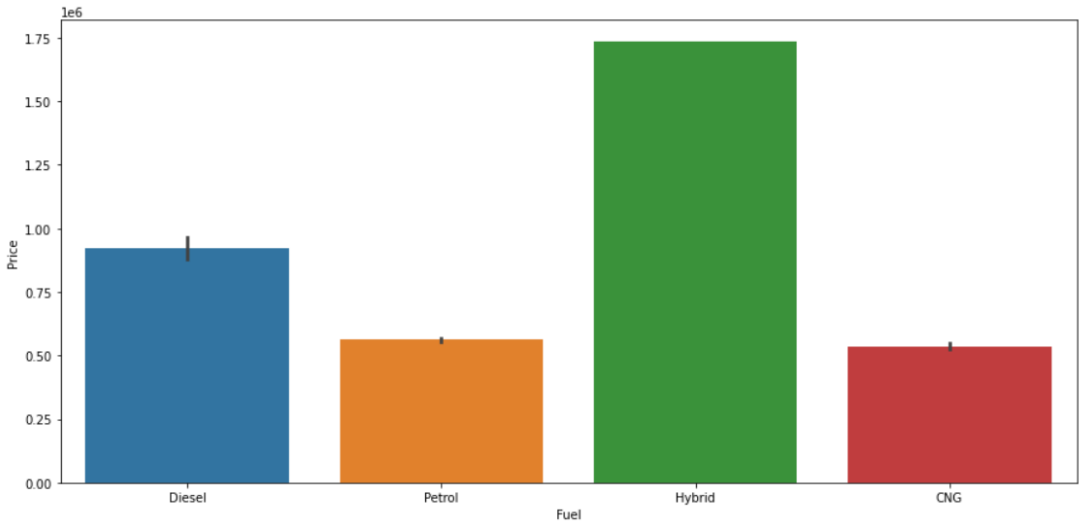
From the above bar graph, we observe there are 13 different locations, and all cars are listed from these locations and mostly cars are listed from Delhi with 500 cars followed by Bangalore, Noida, Ghaziabad with 480 car.



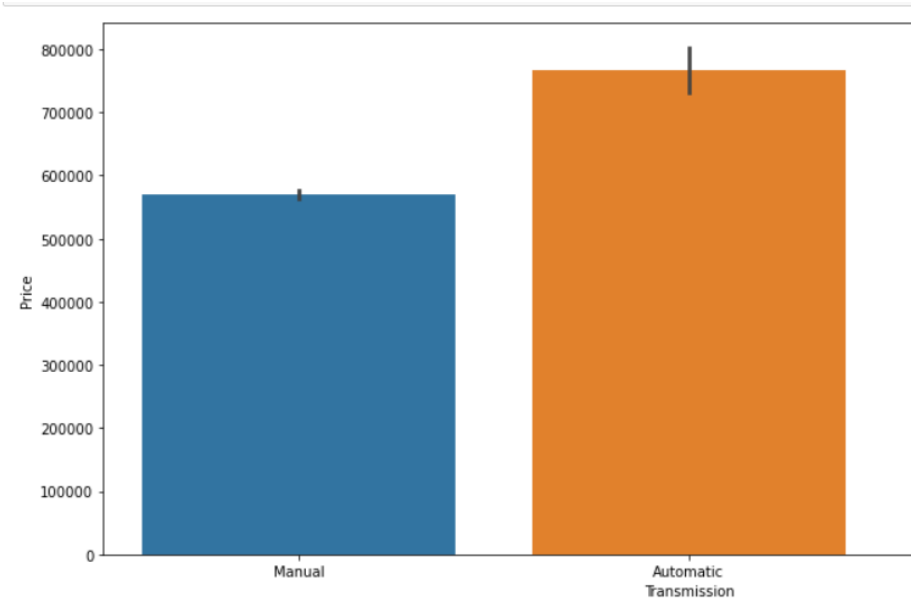
The bar graph is an illustration that shows the comparison of prices among different car brands. The information depicted in the bar graph indicates that the brands Jeep, MG, Maruti, and Audi have a relatively higher price compared to the rest of the car brands.



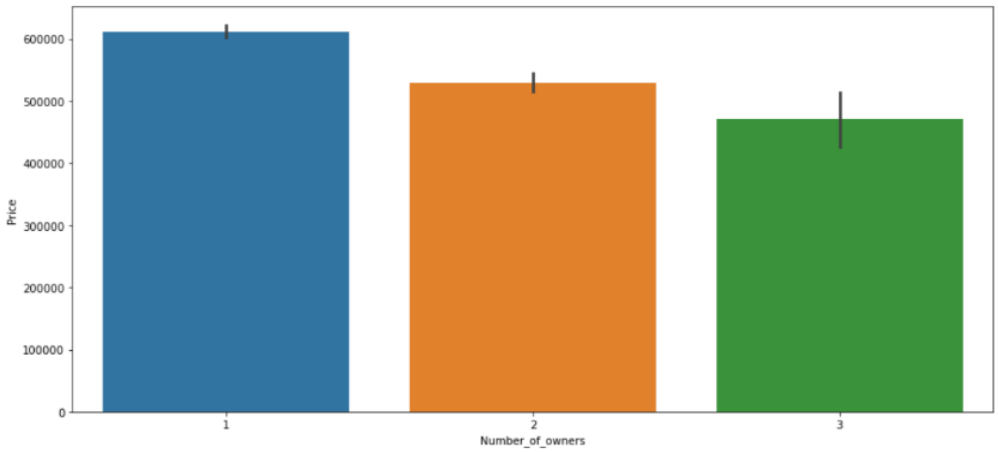
The bar graph provides information and comparison of prices among various cars based on their age. The bar graph reveals that cars which are older tend to have lower prices in comparison to the newest cars available in the market. This implies that the newest car models tend to have higher prices compared to the older models.



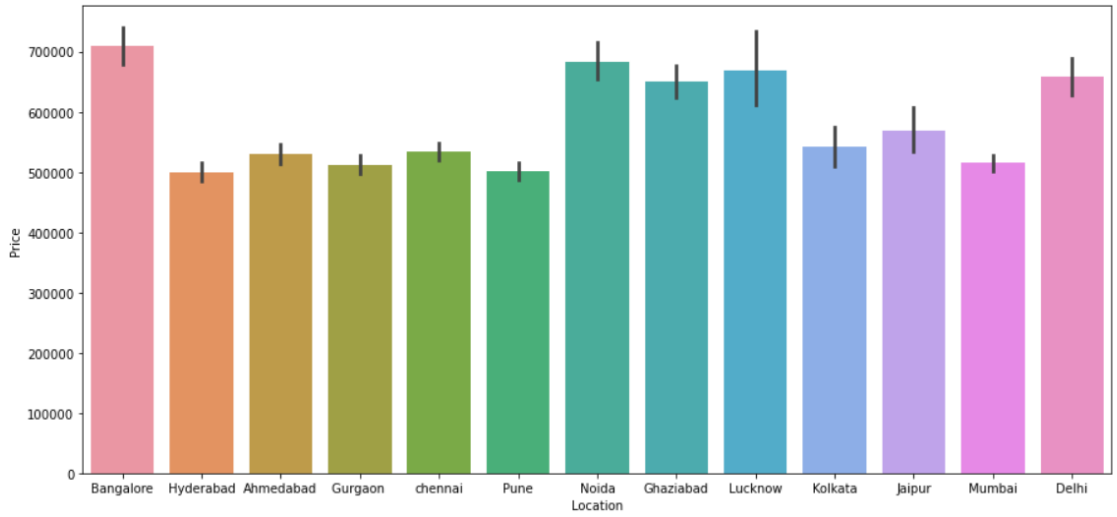
The bar graph presents information about the comparison of prices among different types of fuel-powered cars, including hybrid, diesel, petrol, and CNG cars. As indicated by the bar graph, hybrid cars tend to have the highest price, followed by diesel cars, and then by petrol and CNG cars, which have relatively lower prices compared to diesel and hybrid cars.



The bar graph provides a comparison of prices among cars based on their transmission type, which can either be automatic or manual. The bar graph shows that cars with automatic transmission tend to have a higher price compared to those with manual transmission. This suggests that the type of transmission a car has can greatly impact its overall price.



The bar graph presents a comparison of prices among cars based on the number of owners they have had. The bar graph indicates that cars with one owner tend to have a higher price compared to those with multiple owners. Furthermore, as the number of owners increases, the price of the car tends to decrease, suggesting that the number of owners a car has had can impact its overall price.



The bar graph provides a comparison of prices among cars based on the location where they are listed for sale. The bar graph shows that cars listed in Bangalore, Noida, Ghaziabad, Lucknow, and Delhi tend to have higher prices compared to cars listed in other locations. This suggests that the location where a car is listed for sale can have a significant impact on its overall price.

CONCLUSION

Key Findings and Conclusions of the Study

So from above all 4 model scores, we observe Gradient Boosting Regressor Model is best Suited model for this particular model as the training score is 98.04335166709452 % and the testing score is 90.42506463306986 % thus saving this model.

Learning Outcomes of the Study in respect of Data Science

1) First we look for null values and we treat all null that are present in dataset.

2) Then I identified duplicates and I have dropped duplicates

3) Then we Performed EDA and wrote all observations for each graph

4) Then I dropped unnecessary columns

5) Then applied a label encoder to the categorical columns

6) Then also plotted the Distribution plot and regression plot

7) Then plotted boxplot to remove outliers

8) Then treated outliers with the Z-score method

9) Then scaled data and Also check for VIF

10) Then find the co-relation between feature and label by the CORR method.

11) Then we selected all the features because VIF was lesser than 5.

12) Then I created 4 models that are Gradient Boosting Regressor, Random Forest Regressor, linear Regressor model, and Ada Boosting regressor model with hyperparameter tuning for all 4 models.

13)At last I selected the best model according to their Hyperparameter score and Training(R2) and testing score(R2)

Limitations of this work and Scope for Future Work

Limitations of a car price prediction model in data science can include:

- Data availability: The model may not be able to make accurate predictions if the data used for training is outdated, limited, or biased.
- Feature selection: The model may not perform well if the wrong features are selected or if important features are omitted.
- Model complexity: A highly complex model may overfit the data, leading to poor generalization performance.
- Model evaluation: The accuracy of the model can be influenced by the method used for evaluating its performance, such as the use of cross-validation or a hold-out test set.

Scope for future work in car price prediction could include:

- Improving data quality: Acquiring and incorporating more diverse, recent, and accurate data into the model to improve its predictions.
- Enhancing feature engineering: Creating new features or improving existing ones to better capture important information about the cars.
- Model ensemble: Combining multiple models to create a more robust and accurate prediction.
- Incorporating domain knowledge: Incorporating expert knowledge about the car market into the model to improve its predictions.
- Real-time updates: Updating the model in real-time to reflect changes in the market and improve its predictions.