Car Price Prediction

Submitted by:

PRATIK KUMAR

CAR PRICE PREDICTION

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:

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.

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 model **Conceptual Background of the Domain Problem:**

Linear Regression: Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

Naming the Variables. There are many names for a regression's dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressed. The independent variables can be called exogenous variables, predictor variables, or regressors.

Three major uses for regression analysis are (1) determining the strength of predictors, (2) forecasting an effect, and (3) trend forecasting.

First, the regression might be used to identify the strength of the effect that the independent variable(s) have on a dependent variable. Typical questions are what is the strength of relationship between dose and effect, sales and marketing spending, or age and income.

Second, it can be used to forecast effects or impact of changes. That is, the regression analysis helps us to understand how much the dependent variable changes with a change in one or more independent variables. A typical question is, "how much additional sales income do I get for each additional $1000 spent on marketing?"

Third, regression analysis predicts trends and future values. The regression analysis can be used to get point estimates. A typical question is, "what will the price of gold be in 6 months?"

Types of Linear Regression:

Simple Linear Regression: 1 dependent variable (interval or ratio) , 2+ independent variables (interval or ratio or dichotomous).

Logistic Regression: 1 dependent variable (dichotomous), 2+ independent variable(s) (interval or ratio or dichotomous)

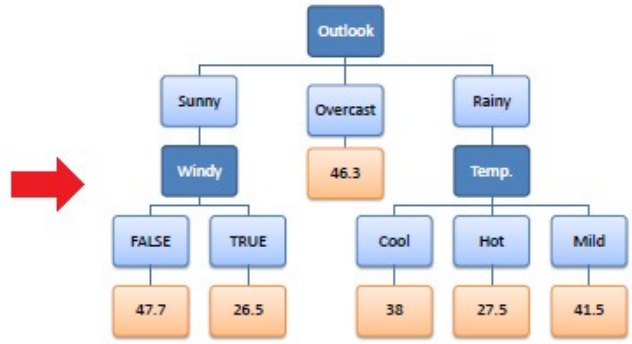
Ordinal Regression:1 dependent variable (ordinal), 1+ independent variable(s) (nominal or dichotomous)

Multinomial Regression: 1 dependent variable (nominal), 1+ independent variable(s) (interval or ratio or dichotomous)

Discriminant Analysis: 1 dependent variable (nominal), 1+ independent variable(s) (interval or ratio)

Decision Tree Regression: Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

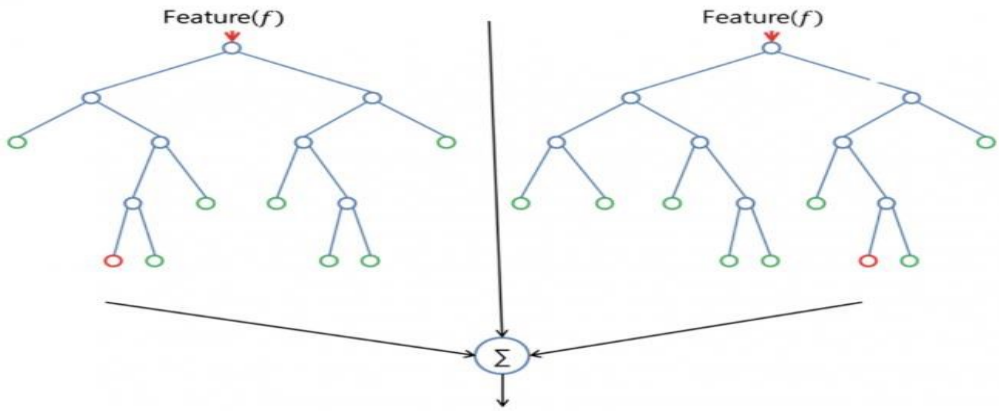
Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Decision Tree Algorithm: The core algorithm for building decision trees called **ID3** by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with *Standard Deviation Reduction*.

Standard Deviation: A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). We use standard deviation to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous its standard deviation is zero.

Random Forest Regression: Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.
Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.
Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Lasso And Ridge Regression: Ridge and Lasso might appear to work towards a common goal, the inherent properties and practical use cases differ substantially. If you’ve heard of them before, you must know that they work by penalizing the magnitude of coefficients of features along with minimizing the error between predicted and actual observations. These are called ‘regularization’ techniques. The key difference is in how they assign penalty to the coefficients:

1. **Ridge Regression:**
    - Performs L2 regularization, i.e., adds penalty equivalent to **square of the magnitude** of coefficients
    - Minimization objective = LS Obj + α * (sum of square of coefficients)
2. **Lasso Regression:**
    - Performs L1 regularization, i.e., adds penalty equivalent to **absolute value of the magnitude** of coefficients o Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

Gradient Boosting Regressor:

Gradient boosting is a technique attracting attention for its prediction speed and accuracy, especially with large and complex data. Don't just take my word for it, the chart below shows the rapid growth of Google searches for xgboost (the most popular gradient boosting R package). From data science competitions to machine learning solutions for business, gradient boosting has produced best-in-class results. In this blog post I describe what is gradient boosting and how to use gradient boosting.

Ensembles and boosting

Machine learning models can be fitted to data individually, or combined in an *ensemble*. An ensemble is a combination of simple individual models that together create a more powerful new model.

Machine learning boosting is a method for creating an ensemble. It starts by fitting an initial model (e.g., a tree or linear regression) to the data. Then a second model is built that focuses on accurately predicting the cases where the first model performs poorly. The combination of these two models is expected to be better than either model alone. Then you repeat this process of boosting many times. Each successive model attempts to correct for the shortcomings of the combined boosted ensemble of all previous models.

Gradient boosting explained

Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error. How are the targets calculated? The target outcome for each case in the data depends on how much changing that case's prediction impacts the overall prediction error:

- If a small change in the prediction for a case causes a large drop in error, then next target outcome of the case is a high value. Predictions from the new model that are close to its targets will reduce the error.

- If a small change in the prediction for a case causes no change in error, then next target outcome of the case is zero. Changing this prediction does not decrease the error.

The name *gradient boosting* arises because target outcomes for each case are set based on the gradient of the error with respect to the prediction. Each new model takes a step in the direction that minimizes prediction error, in the space of possible predictions for each training case.

Data Analysis: The Dataset Contains a Data of 4168 entries each having 11 variables, in which some are numerical Data and some are Categorical Data .

```
In [44]: 1 ## Checking null values:
         2 df.isnull().sum()

Out[44]: LOCATION        0
         MNF_YEAR        0
         BRAND           0
         MODEL           0
         VARIANT         0
         TRANSMISSION    81
         DRIVEN_KM       0
         FUELTYPE        0
         NO_OF_OWNERS    0
         PRICE           0
         dtype: int64
```

Since TRANSMISSION is a categorical column.so filling null values with mode in TRANSMISSION column:

```
In [176]:
## Since  TRANSMISSION is a categorical column.so filling null values with mode in TRANSMISSION column:
df['TRANSMISSION'].fillna(df['TRANSMISSION'].mode()[0],inplace=True)

In [177]:
df.isnull().sum()

Out[177]:
LOCATION        0
MNF_YEAR        0
BRAND           0
MODEL           0
VARIANT         0
TRANSMISSION    0
DRIVEN_KM       0
FUELTYPE        0
NO_OF_OWNERS    0
PRICE           0
dtype: int64
```

Now There is not any null values are present in dataset.

a) During analysis I found that DRIVEN_KM and PRICE have km and Rs. appended before, So I have formatted that and convert their datatype to int.

```
df['DRIVEN_KM'] = df['DRIVEN_KM'].astype(int)
```

```
df['DRIVEN_KM'].dtypes
```

```
df['PRICE'] = pp
df['PRICE'] =df['PRICE'].astype(int)
```

```
df['PRICE'].dtypes
```

```
dtype('int32')
```

b) Encode categorical features using LabelEncoder.

```
## Label Encoding:
columns = ['MNF_YEAR','LOCATION','BRAND','MODEL','VARIANT','TRANSMISSION','FUELTYPE','NO_OF_OWNERS']
lec = LabelEncoder()
for i in columns:
    df[i] = df[i].astype('str')
    df[i]=  lec.fit_transform(df[i])
```

```
df
```

	LOCATION	MNF_YEAR	BRAND	MODEL	VARIANT	TRANSMISSION	DRIVEN_KM	FUELTYPE	NO_OF_OWNERS	PRICE
0	7	8	12	94	497	1	101529	0	0	655999
1	7	8	12	94	424	1	59773	0	0	666399
2	7	9	12	94	424	1	35231	0	0	684399
3	7	5	4	30	101	1	80946	0	1	409899
4	7	8	4	30	86	1	16212	1	0	741699
...	...	...	...	...	...	...	...	...	...	...
8663	0	11	12	31	172	1	26880	1	0	459599
8664	0	11	12	31	172	1	11882	1	0	461999
8665	0	9	4	30	101	1	71900	0	1	616799
8666	0	10	12	94	419	1	27460	0	0	816199
8667	0	12	4	30	101	1	15951	0	0	1039699

8668 rows × 10 columns

c) Checked correlation using heatmap.

Correlation

```
plt.figure(figsize =(30,30))
sns.heatmap(df.corr(),annot = True,cmap ='RdYlGn')
plt.show()
```

d) Detected outliers using **distplot** and **boxplot.**

e) Removed skewness using Power Transformer, method is ‘yeo-johnson’.

```
df.skew()
```

```
LOCATION       -0.403131
MNF_YEAR       -0.462373
BRAND           0.190042
MODEL           0.241976
VARIANT        -0.330499
TRANSMISSION   -1.594060
DRIVEN_KM      -1.402540
FUELTYPE       -0.220277
NO_OF_OWNERS    2.083213
PRICE           1.811769
dtype: float64
```

```
#Except DRIVEN_KM and PRICE ,all are categorical columns. PRICE is target variable,so will remove skewness from DRIVEN_KM.
from sklearn.preprocessing import PowerTransformer
skewed_features = ['DRIVEN_KM']
scaler = PowerTransformer(method='yeo-johnson')
```

```
df[skewed_features] = scaler.fit_transform(df[skewed_features].values)
df[skewed_features].head()

```

	DRIVEN_KM
0	1.400642
1	0.552685
2	-0.163707
3	1.021060
4	-1.012395

f) Separated Features and Target

g) Standardize data using Standard Scaler.

Generalizing data using Standard Scaler

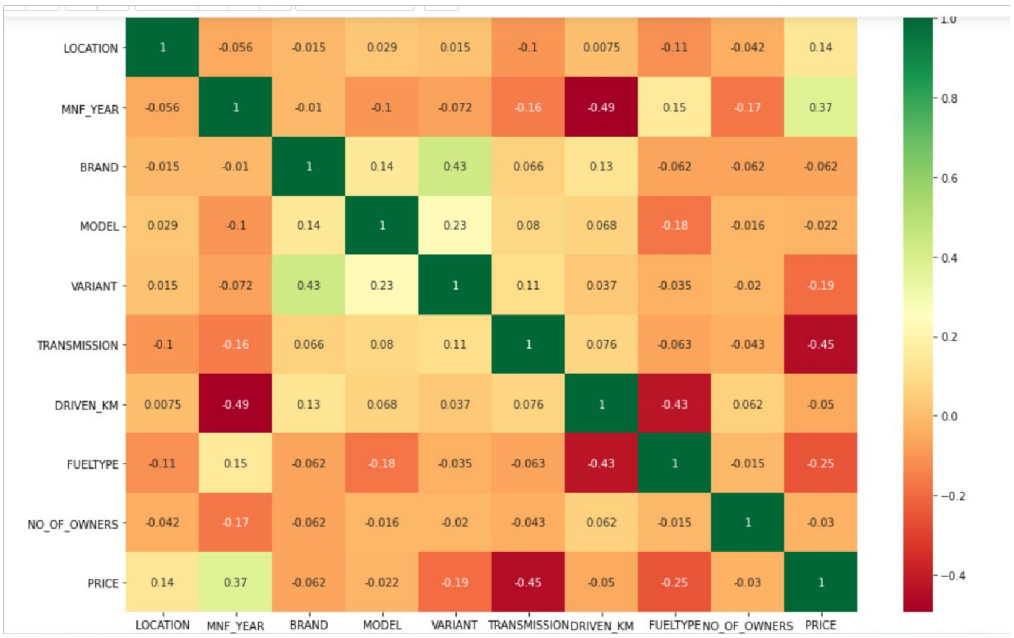
```
scaler = StandardScaler()
x=pd.DataFrame(scaler.fit_transform(x),columns=x.columns)
x
```

	LOCATION	MNF_YEAR	BRAND	MODEL	VARIANT	TRANSMISSION	DRIVEN_KM	FUELTYPE	NO_OF_OWNERS
0	0.536716	-0.152979	0.447871	1.324209	1.264121	0.481794	1.400642	-1.214091	-0.469389
1	0.536716	-0.152979	0.447871	1.324209	0.793740	0.481794	0.552685	-1.214091	-0.469389
2	0.536716	0.291995	0.447871	1.324209	0.793740	0.481794	-0.163707	-1.214091	-0.469389
3	0.536716	-1.487899	-1.337846	-0.687676	-1.287537	0.481794	1.021060	-1.214091	1.743043
4	0.536716	-0.152979	-1.337846	-0.687676	-1.384191	0.481794	-1.012395	0.754435	-0.469389
...	...	...	...	...	...	...	...	...	...
8663	-1.894292	1.181941	0.447871	-0.656240	-0.830043	0.481794	-0.485204	0.754435	-0.469389
8664	-1.894292	1.181941	0.447871	-0.656240	-0.830043	0.481794	-1.292932	0.754435	-0.469389
8665	-1.894292	0.291995	-1.337846	-0.687676	-1.287537	0.481794	0.832749	-1.214091	1.743043
8666	-1.894292	0.736968	0.447871	1.324209	0.761522	0.481794	-0.460878	-1.214091	-0.469389
8667	-1.894292	1.626915	-1.337846	-0.687676	-1.287537	0.481794	-1.027831	-1.214091	-0.469389

8668 rows × 9 columns

Data Inputs- Logic- Output Relationships

Following is the Heatmap which shows inputs -output Relationship:



Hardware and Software Requirements and Tools Used

	PRICE
MNF_YEAR	0.373988
LOCATION	0.139629
MODEL	-0.021679
NO_OF_OWNERS	-0.030471
DRIVEN_KM	-0.050042
BRAND	-0.062152
VARIANT	-0.194721
FUELTYPE	-0.253148
TRANSMISSION	-0.447728

MNF Year and Location have positive relationship with Target. Model,No of owners,driven km,brand, variant, Fuel Type,Transmission have negative co with Target.

Hardware:

- Process: Intel core i5 and above.
- RAM:16GB and above.
- SSD:250GB and above.

Software & Tools used:

- Python3.0,
- Jupyter Notebook

Libraries:

Pandas: pandas is a popular Python-based data analysis toolkit which can be imported using import pandas as pd. It presents a diverse range of utilities, ranging from parsing multiple file formats to converting an entire data table into a numpy matrix array. This makes pandas a trusted ally in data science and machine learning.

Numpy: NumPy is the fundamental package for package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation,

sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

• **Seaborn**: Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas' data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data.

• **Matplotlibt**: matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some linesin a plotting area, decorates the plot with labels, etc.

ι. from sklearn.preprocessing import StandardScaler
ιι. from sklearn.preprocessing import PowerTransformer
ιιι. from sklearn.tree import DecisionTreeRegressor
ιϖ. from sklearn.ensemble import RandomForestRegressor
ϖ. from sklearn.linear_model import LinearRegression,Lasso,LassoCV,Ridge,RidgeCV
ϖι. from sklearn.ensemble import AdaBoostRegressor
ϖιι. from sklearn.neighbors import KNeighborsRegressor
ϖιιι. from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
ιξ. from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error ## to get score of our model.
ξ. from sklearn.preprocessing import LabelEncoder ## To encode our categorical variable into numerical variable.
ξι. from sklearn.preprocessing import StandardScaler

Model/s Development and Evaluation

Identification of possible problem-solving approaches: Since this is a regression problem, So I have first tried to find best random states for my regression models.

```
#model building
```

```
score = 0
for i in range(0,200):
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = .30, random_state = i)
    rf = RandomForestRegressor()
    rf.fit(x_train,y_train)
    y_pred = rf.predict(x_test)
    tempscore = r2_score(y_test,y_pred)
    if tempscore > score:
        score= tempscore
        best_rstate=i

print(f"Best Accuracy {score*100} found on randomstate {best_rstate}")
```

```
Best Accuracy 98.55411073750815 found on randomstate 47
```

Finding Best random state

b) Initialize models and train our models and then predict our test target data and checked all metrics ,so that I can know that how my models are performing and which is best for this data set.

```
lr=LinearRegression()
dtr=DecisionTreeRegressor()
knn=KNeighborsRegressor()
rf=RandomForestRegressor()
AdaBoost=AdaBoostRegressor()
lasso = Lasso()
ridge  = Ridge(alpha=1, random_state=42)
```

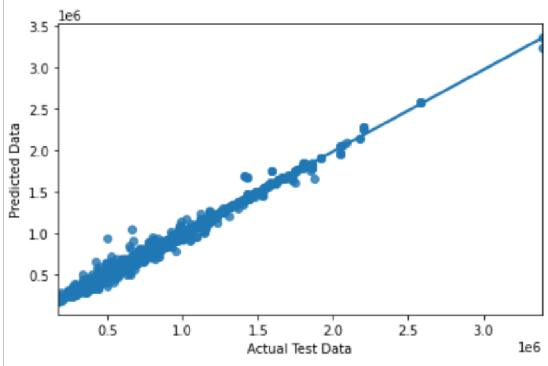
```
algo=[lr,dtr,rf,knn,AdaBoost,lasso,ridge]
acc_models={}
for model in algo:
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
    print("-"*60)
    acc_models[model]=round(r2_score(y_test,y_pred)*100,1)
    print(f"The model {model} has:: \n\t Accuracy :: {round(r2_score(y_test,y_pred)*100,1)}% \n\t Mean Absolute Error is ::
    print("-"*100)
    print("\n")
```

Run and evaluate selected models

Regplot for Random Forest Model:

```
# Plotting regplot graph for Random Forest Regressor model
```

```
rf=RandomForestRegressor()
rf.fit(x_train,y_train)
y_pred=rf.predict(x_test)
sns.regplot(y_test,y_pred)
plt.xlabel("Actual Test Data")
plt.ylabel("Predicted Data")
plt.tight_layout()
```



Here we can see that the data points are very close to the best fit line. That means the residual is less.So I'll do hyperparameter tuning for Random Forest.

g) HyperParameter Tuning of the model:

```
param_grid = [
    {"bootstrap": [True, False],
     "criterion": ["mse", "mae"],
     "n_estimators"      : [10,20,30,50,100],
     "max_features"      : ["auto", "sqrt", "log2"],
     "min_samples_split" : [2,4,8],
    }
]

rf = RandomForestRegressor(random_state=best_rstate)

grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 2)

grid_search.fit(x_train,y_train)
final_model = grid_search.best_estimator_
final_model
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits

RandomForestRegressor(bootstrap=False, max_features='sqrt', min_samples_split=4,
                      random_state=47)
```

```
## Providing hyperparameters to model:-
rfr = RandomForestRegressor(bootstrap=False,max_features='sqrt', random_state=42 )
rfr.fit(x_train, y_train)
y_pred = rfr.predict(x_test)
score =  r2_score(y_test,y_pred)
score
```

```
After Hyperparameter tuning score has increased.
```

CONCLUSION

```
a = np.array(y_test)
predicted = np.array(rfr.predict(x_test))
new_df = pd.DataFrame({"Original":a,"Predicted":predicted},index= range(len(a)))
new_df
```

	Original	Predicted
0	443499	437759.0
1	564499	564499.0
2	413799	413799.0
3	274799	292299.0
4	1176099	1176099.0
...	...	...
2200	767999	766297.0
2201	465699	465699.0
2202	873999	873999.0
2203	1030399	1030399.0
2204	267799	267799.0

2205 rows × 2 columns

The above model will help our seller to predict the Price of the used cars, and also will helps them to understand based on factors that affects used car's pricing.

Concluding Remarks: - From this model we can predict the Car price prediction of different variables how the prices are varying according to that each one can take their prescribed house.