Car Price Prediction Project

Submitted By

NAROJ SANTHOSH 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)

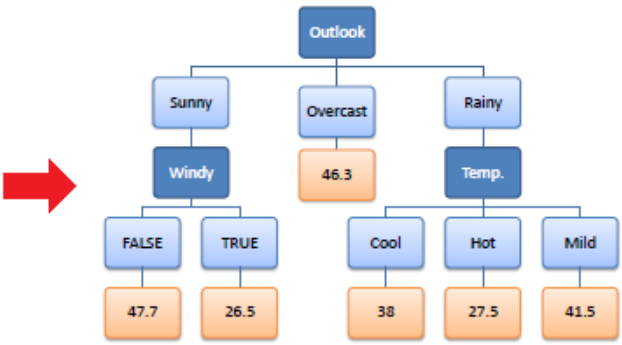
Ordinal Regression:1 dependent variable (ordinal), 1+ independent variable(s) (nominal or dichotomous)

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.

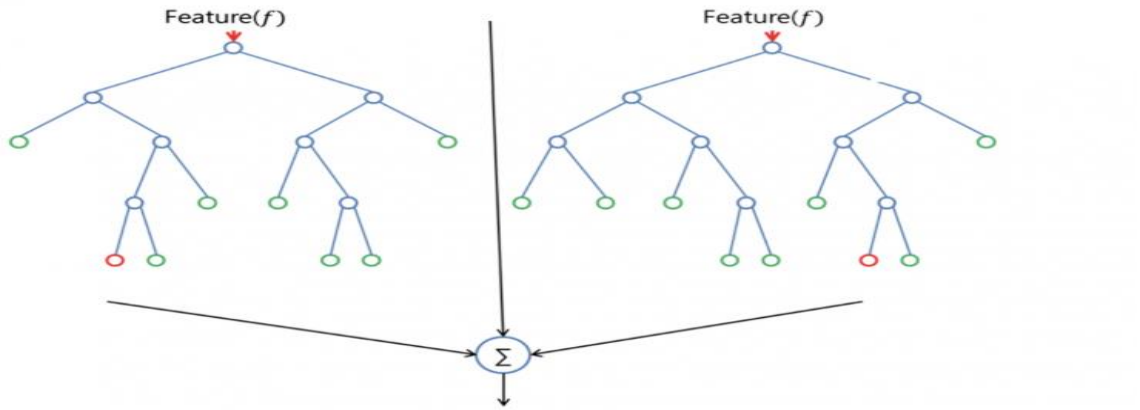
Predictors				Target
Outlook	Temp	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
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
    - 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

```
: 1 df['company name'].value_counts()
```

```
: Maruti          2267
  Hyundai          916
  Ford             251
  Honda            189
  Renault           99
  Volkswagen        86
  Toyota            77
  Tata              66
  Mahindra          39
  KIA               34
  Skoda             28
  MG                28
  Datsun            25
  Audi              15
  Jeep              13
  BMW               11
  Mercedes          10
  Ssangyong          8
  Nissan             4
  HYUNDAI            1
  RENAULT            1
  Name: company name, dtype: int64
```

As the Above Data-set we are having different car company names counts

```
1 df.drop('Variant',inplace=True,axis=1)
```

```
1 df.describe()
```

	Unnamed: 0	year
count	4168.00000	4168.000000
mean	2083.50000	2016.991123
std	1203.34229	2.231538
min	0.00000	2008.000000
25%	1041.75000	2016.000000
50%	2083.50000	2017.000000
75%	3125.25000	2019.000000
max	4167.00000	2021.000000

We removed the variant which it will not provide any impact on our Data and describe function gives how many are numerical and their data values of mean, median etc., Here we have only one column as int that is year. and we have oldest model of the year 2008 and latest model is 2021.

```
tabel=[]

for i in df.columns:
    tabel.append([i,df[i].nunique(),df[i].drop_duplicates().values])

pd.DataFrame(tabel,columns=['Features','Unique No','Values'])
```

	Features	Unique No	Values
0	Unnamed: 0	4168	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
1	year	14	[2009, 2020, 2018, 2016, 2015, 2011, 2014, 201...
2	company name	19	[Toyota, Hyundai, Maruti, Mahindra, Volkswagen...
3	car model	107	[Camry, VENUE, Swift, Kuv100, Creta, Etios, Am...
4	km driven	2189	[99,864, 4,489, 547, 8,339, 44,486, 1,10,584, ...
5	ownership	4	[1st Owner, 2nd Owner, 3rd Owner, 4th Owner]
6	Fuel	4	[Petrol, Diesel, Petrol + CNG, Petrol + LPG]
7	Price	1923	[₹3,07,299, ₹9,52,499, ₹8,26,399, ₹5,86,999, ₹...
8	Transmission	2	[Automatic, Manual, nan]
9	City	30	[Mumbai, New Delhi, Gurgaon, Pune, Bengaluru, ...

As Our Above one shows us the features of our data and their unique values and removed the duplicates.

```
df.isnull().sum()
```

```
Unnamed: 0       0
year             0
company name     0
car model        0
km driven        0
ownership        0
Fuel             0
Price            0
Transmission    95
City             0
dtype: int64
```

It shows that one of our column having 95 null values need to change by using different methods

Exploratory Data Analysis:

In EDA we need to Pre-process the Data and Visualization:

Steps include in Pre-Processing Data are

1)**Data Cleaning**: - Removing Outliers, Skewness and imputing Missing Values.

2)**Data Transformation**: - like Normalization by applying normalization we can improve the accuracy and efficiency of the models. And also reduce the errors.

3)**Data Reduction**: By Reducing the no of features by Feature Selection Process, PCA And VIF

1.Data Cleaning: As a Part of EDA, we need to do Data cleaning so firstly we need to check any null values in our data, From the below image shows we don't have any null values, so no need to impute any data

```
1 df['km driven']=df['km driven'].astype('int64')
2 df['Price']=df['Price'].astype('int64')
```

```
1 df.info()
```

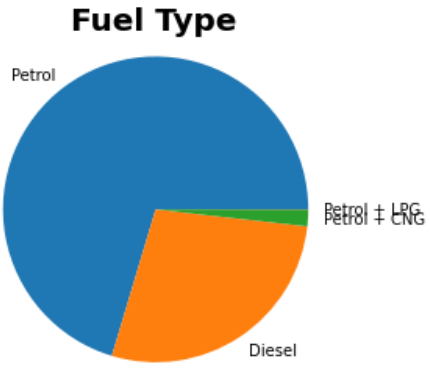
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4168 entries, 0 to 4167
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    4168 non-null   int64
 1   year          4168 non-null   int64
 2   company name  4168 non-null   object
 3   car model     4168 non-null   object
 4   km driven     4168 non-null   int64
 5   ownership     4168 non-null   object
 6   Fuel          4168 non-null   object
 7   Price         4168 non-null   int64
 8   Transmission  4073 non-null   object
 9   City          4168 non-null   object
dtypes: int64(4), object(6)
memory usage: 325.8+ KB
```

In this we are changing the km and price into integer type

VISUALIZATION:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
plt.pie(df['Fuel'].value_counts(),labels=df['Fuel'].value_counts().index)
plt.title('Fuel Type',fontdict={'fontweight':'bold','fontsize':20})
plt.axis('equal')
plt.show()
```



There are different Fuel Types mostly people are okay with the Petrol Cars Compare to other vehicles

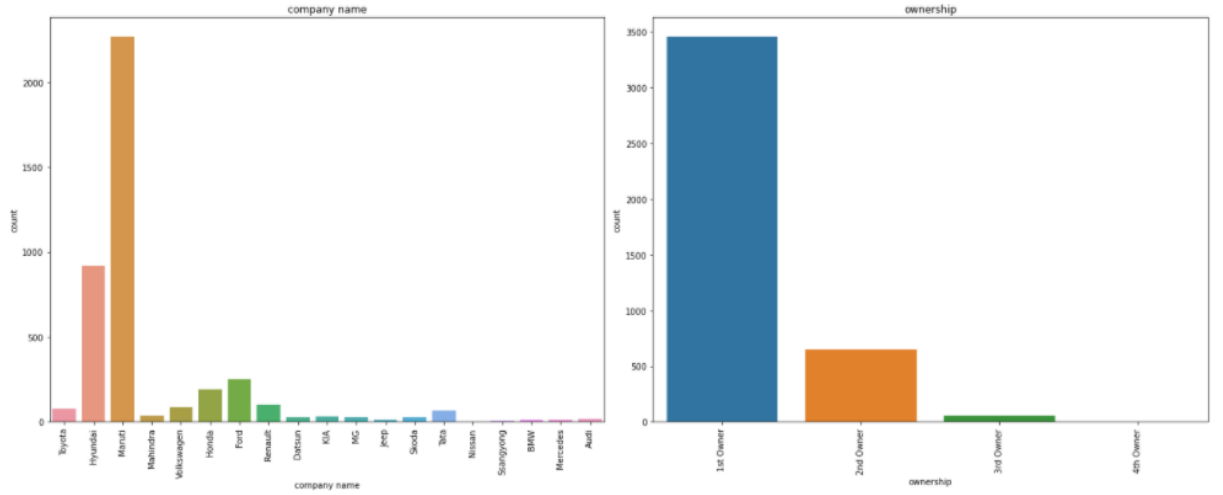
```
cat=[feature for feature in df.columns if df[feature].dtype=='O']
cat
```

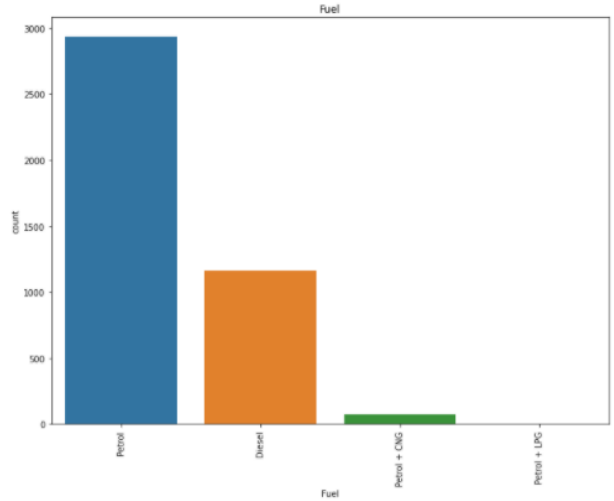
```
['company name', 'car model', 'ownership', 'Fuel', 'Transmission', 'City']
```

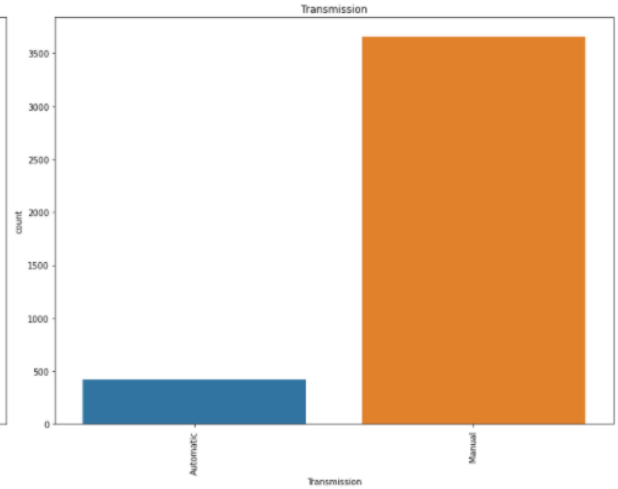
```
col=['company name', 'ownership', 'Fuel', 'Transmission', 'City']
```

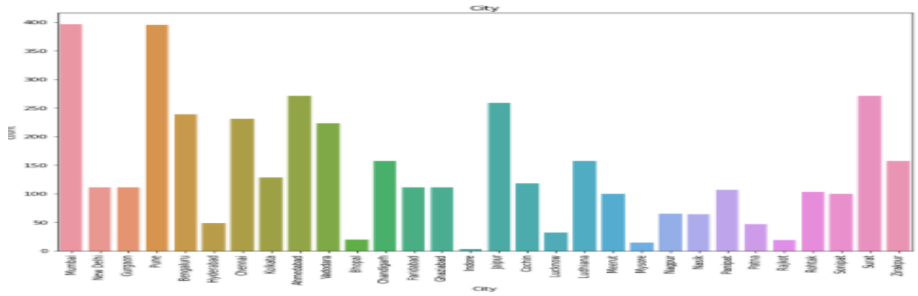
These are the categorical Data types in our Data.

```
plt.figure(figsize=(20,40))
for i in range(len(col)):
    plt.subplot(5,2,i+1)
    sns.countplot(df[col[i]])
    plt.title(col[i])
    plt.xticks(rotation=90,fontsize=10)
    plt.tight_layout()
```









Observation

1)Maruti having high count for sell as compare to other brans which means that people who used Maruti's car they sell it for second Hand. after Maruti, Honda and Hyundai are sold second hand.

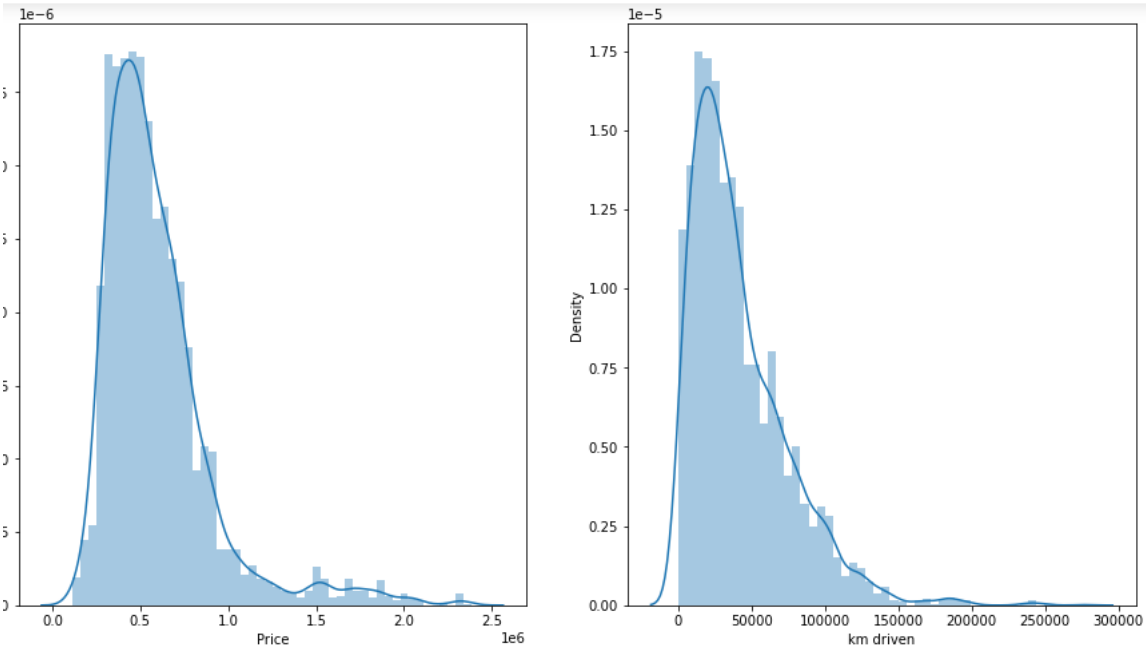
2)Most of the cars are from the 1st owner and after that 2nd and 3rd owners.

3)Most of the people are prefer the petrol cars because they have low militance cost as compare to Diesel and life of petrol car is high than the Diesel car and some people use the Diesel car because they have good mileage.

4)Most of the cars are Manually Transmission and very few are Automatic. Because Automatic having high cost as compare to Manual

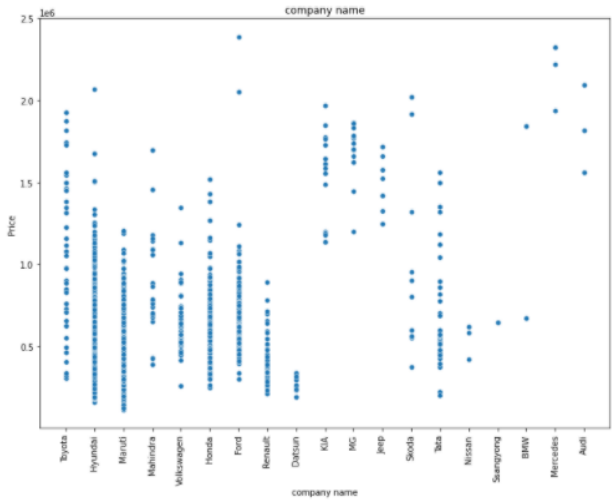
5)They are many people from Mumbai, Pune, Jaipur and Surat sell their car and from city like Rajkot, Chennai, Hyderabad, Indore they are very less for selling the cars.

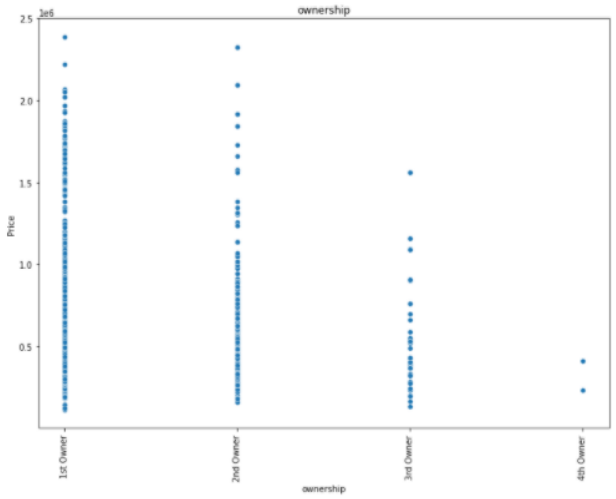
```
plt.figure(figsize=(15,8))
plt.subplot(1,2,1)
sns.distplot(df['Price'])
plt.subplot(1,2,2)
sns.distplot(df['km driven'])

```

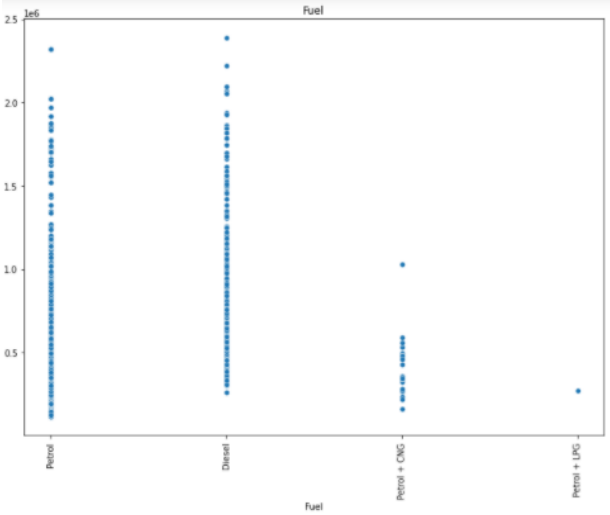


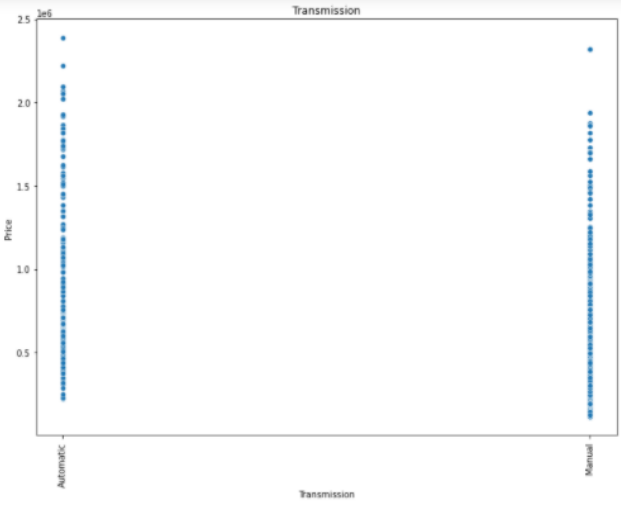
From the charts we know that KM and Price are skewed

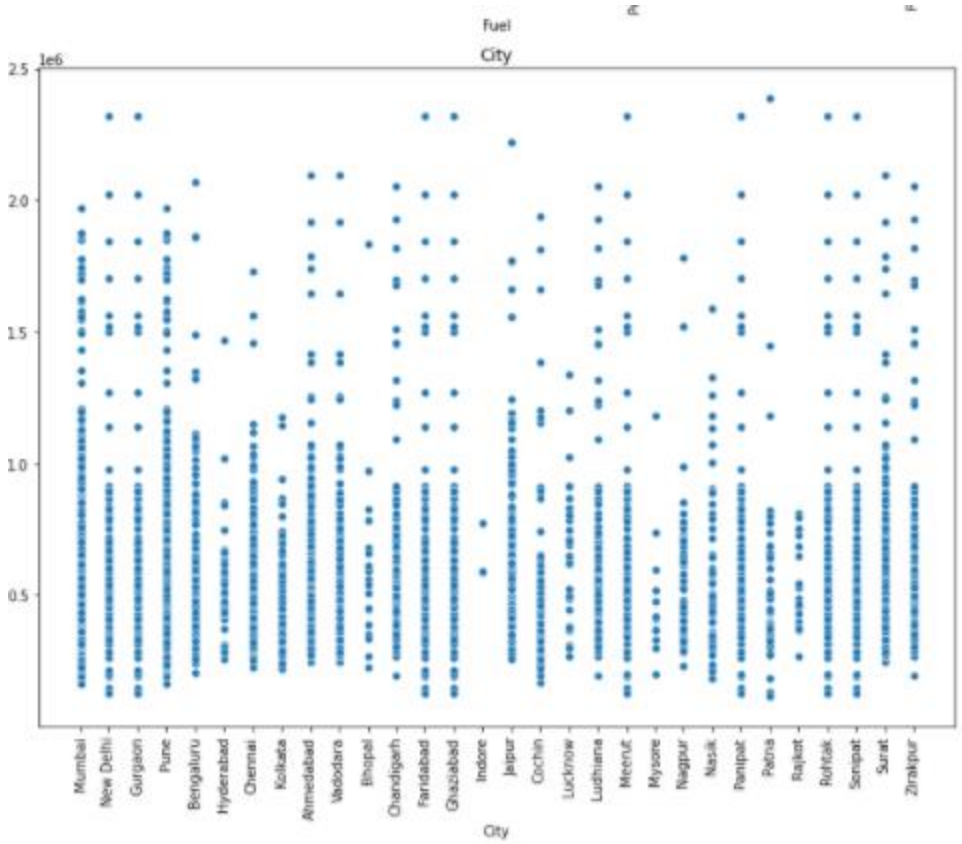
```
plt.figure(figsize=(20,40))
for i in range(len(col)):
    plt.subplot(5,2,i+1)
    sns.scatterplot(x=df[col[i]],y=df['Price'])
    plt.title(col[i])
    plt.xticks(rotation=90,fontsize=10)
    plt.tight_layout()
```











Observation

1)Company like BMW, Audi, Mercedes having high price after that Jeep, KIA are coming for higher price. Maruti, Toyota, Honda and Tata having relatively low prices.

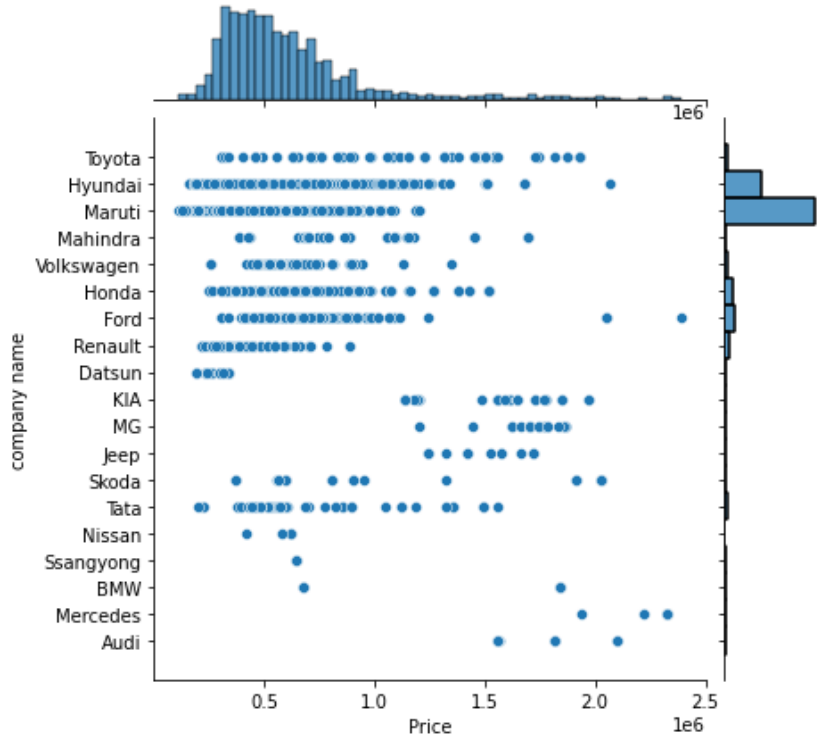
2) Price for the 1st owner is high and after than 2nd owner and follow by 3rd and 4th owner. because as the owner changes chances of low milage and militances cost will increases.

3)price for the petrol is high than petrol+LPG and Petrol+CNG

4) As we easily show that Mumbai, Pune, Surat having high no. of car selling so price is also high for that city as compare to another city like Hyderabad, Mysore, Chennai, Rajkot.

```
1 sns.jointplot(y=df['company name'],x=df['Price'])
2 plt.xticks(rotation=90)
```

```
(array([   0., 2000., 4000.]), [])
```



From these charts we can see that Price of the Maruti is low and volume of the Maruti car is high. and price for the Audi, BMD Mercedes is high and volume is too much low.

Label Encoding:

Label Encoding is necessary for the data to process to find any outliers are there as of our data consists of both numerical and categorical need to change categorical into numerical values using the encoding methods.

```
1 df.drop('Unnamed: 0',axis=1,inplace=True)
```

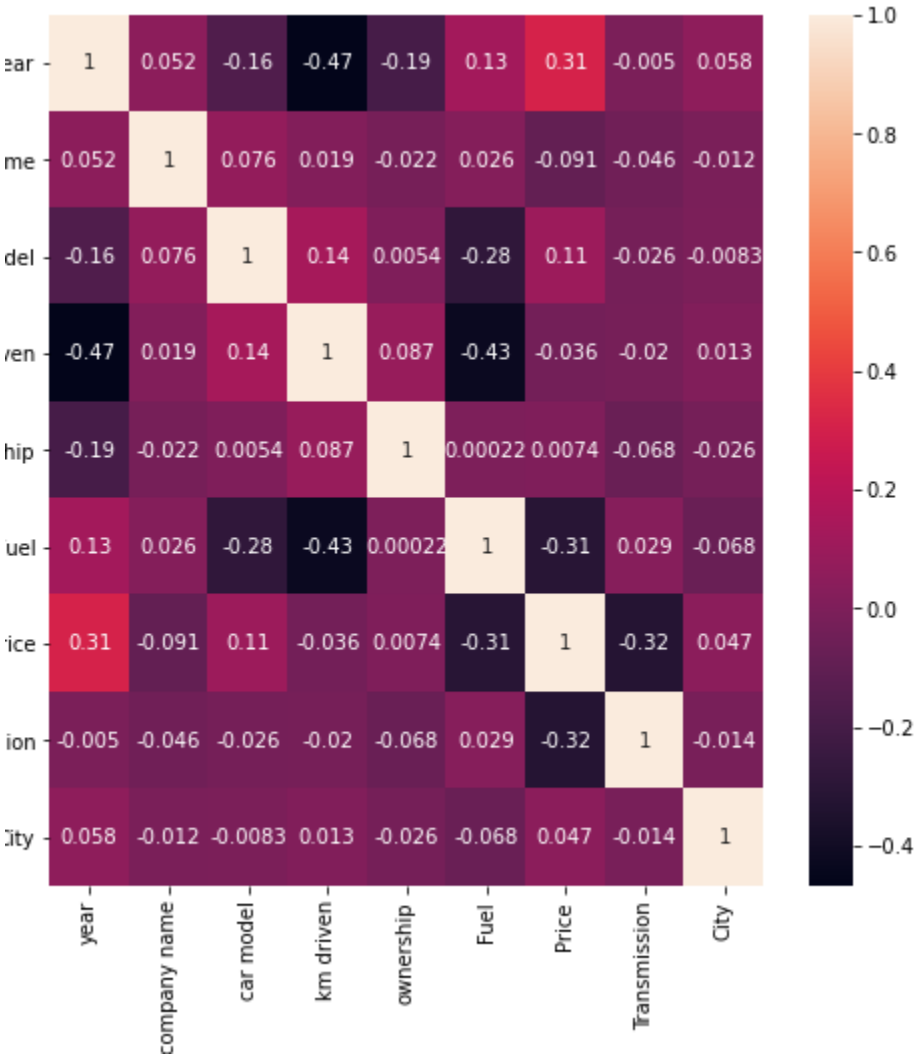
```
1 from sklearn.preprocessing  import LabelEncoder
```

```
1 lb=LabelEncoder()
```

```
1 for i in cat:
2     df[i]=lb.fit_transform(df[i])
```

```
1 df
```

```
plt.figure(figsize=(8,8))
corr=df.corr()
sns.heatmap(data=corr,annot=True)
```



As per the describe analysis, need to drop some more variables to make our model is accurate

Checking Any Outliers in our Data and Remove it:

It is defined as the points that are far away from the same points.it can be happen because of the variability of the measurements and may be some error also. If possible, outliers should be removed from the datasets. There are servals methos to remove the outliers. 1)Z score 2) Quantile Method (Capping the data)

1)Z Score: it can call from the SciPy. Stats library. And for most of the case threshold values should be used 3.

As above shows our data having the outliers so need to remove outlier by one of the prominent methods called z-score method

```
from scipy.stats import zscore
```

```
z=abs(zscore(df))
```

```
threshold=3
print(np.where(z>3))
```

```
df_new=df[(z<3).all(axis=1)]
```

```
df_new.shape
```

```
(3842, 9)
```

```
dataloss=(4059-3741)/100
dataloss
```

```
3.18
```

```
df=df_new
```

Almost 3.18% of our train data and 11% of test data is removed

2)Quantile Methods: Inter Quantile Range is used to detect or cap the outliers. Calculate the IQR by scipy.stats.iqr Multiply Interquartile range by 1.5 Add 1.5 x interquartile range to the third quartile. Any number greater than this is a suspected outlier. Subtract 1.5 x interquartile range from the first quartile. Any number lesser than this is a suspected outlier.

Now our Data is ready for modelling as our data is clean so only thing need to do is normalizing the data before need to check our dependent variable.

```
df.skew()
```

```
year           -0.644051
company name    0.279023
car model       0.334263
km driven       0.969103
ownership       1.884206
Fuel           -0.649749
Price           1.236245
Transmission   -2.945106
City           -0.044608
dtype: float64
```

```
df['km driven']=np.sqrt(df['km driven'])
df['Price']=np.sqrt(df['Price'])
```

```
df
```

```
y=df['Price']

```

```
x=df.drop('Price',axis=1,inplace=True)
x
```

```
x=df
```

```
x
```

Skewness of Data:

As of our numeric data is skewed, we need to do normalization before go for training and testing for that need to check the skewness of data if our data is Greater than 0.5% in both positive and negative sides, then need to do power transformation and do scaling

Scaling is of two types:

1.**Standard Scaler:** Standard scalar standardizes features of the data set by scaling to unit variance and removing the mean (optionally) using column summary statistics on the samples in the training set.

MIN-MAX Scaler: MinMaxScaler. For each value in a feature, MinMaxScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum. MinMaxScaler preserves the shape of the original distribution.

```
1 from sklearn.preprocessing import StandardScaler
```

```
1 std=StandardScaler()
```

```
1 X=std.fit_transform(x)
```

Splitting Data Into train_test_split: -

This function is in sklearn. Model selection splitting the data array into two arrays. Train and Test with this function we don't need to splitting train and test manually.by default it make random partition and we can also set the random state.it gives four o/p like x_train, x_test, y_train, y_test.

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.model_selection import train_test_split,KFold,cross_val_score
3 from sklearn.metrics import mean_squared_error,mean_absolute_error,accuracy_score,r2_score
```

```
1 x_train,x_test,y_train,y_test=train_test_split(X,np.log(y),random_state=42,test_size=0.30)
```

```
1 ln=LinearRegression()
2 ln.fit(x_train,y_train)
3 pred=ln.predict(x_test)
4 print('Taining Score:-',ln.score(x_train,y_train)*100)
5 print('Mean Absolute Error',mean_absolute_error(y_test,pred))
6 print('Mean Squared Error',mean_squared_error(y_test,pred))
7 print('Root Mean Squared Erro',np.sqrt(mean_squared_error(y_test,pred)))
8 print('R2 Score',r2_score(y_test,pred)*100)
9 print("Test Score",ln.score(x_test,y_test)*100)
```

```
Taining Score:- 46.13121055520772
Mean Absolute Error 0.1133657865400659
Mean Squared Error 0.01950195706778023
Root Mean Squared Erro 0.13964940768861223
R2 Score 46.7695636471795
Test Score 46.7695636471795
```

As we do split the data for training and testing the data then now, we need to modeling

Try Different Models….

1)**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.

I simply used function method, so no need to write the code each and every time, just call the model

```
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
import scikitplot as skplt
```

```
svr=SVR()
svrl=SVR(kernel='linear')
svrp=SVR(kernel='poly')
dtc=DecisionTreeRegressor()
knn=KNeighborsRegressor()
```

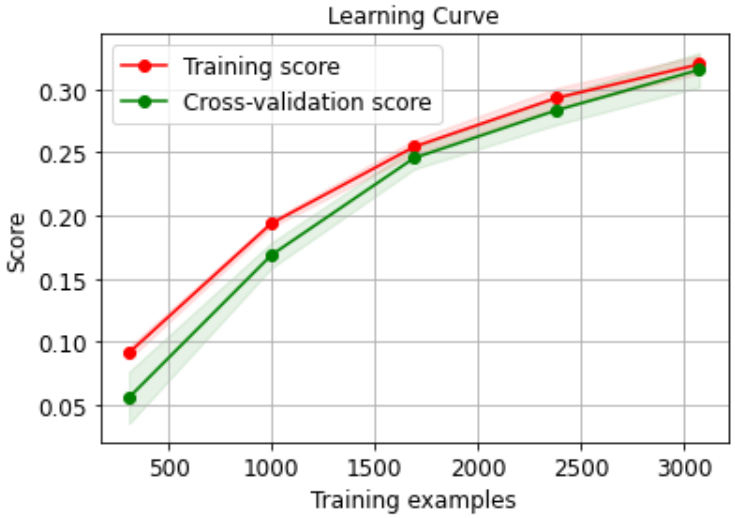
```
kf=KFold(n_splits=5,shuffle=True)
def fun(f):
    f.fit(x_train,y_train)
    pred=f.predict(x_test)
    predt=f.predict(x_train)
    print('----',f,'----')
    print('Taining Score:-',f.score(x_train,y_train)*100)
    print('Mean Absolute Error',mean_absolute_error(y_test,pred))
    print('Mean Squared Error',mean_squared_error(y_test,pred))
    print('Root Mean Squared Error',np.sqrt(mean_squared_error(y_test,pred)))
    print('Cross Validation Score',cross_val_score(f,x,y,cv=kf).mean()*100)
    r2=r2_score(y_test,pred)
    print('R2 Score',r2_score(y_test,pred))
    print("Test Score",f.score(x_test,y_test)*100)
    print('Model Peformance Cure')
    skplt.estimators.plot_learning_curve(f,X,y,cv=kf,scoring='r2',text_fontsize='large')
    plt.show()
```

Support Vector Machine: Support Vector Machines (SVM) are popularly and widely used for classification problems in machine learning. I've often relied on this not just in machine learning projects but when I want a quick result in a hackathon.

But SVM for regression analysis? I hadn't even considered the possibility for a while! And even now when I bring up "Support Vector Regression" in front of machine learning beginners, I often get a bemused expression. I understand – most courses and experts don't even mention Support Vector Regression (SVR) as a machine learning algorithm.

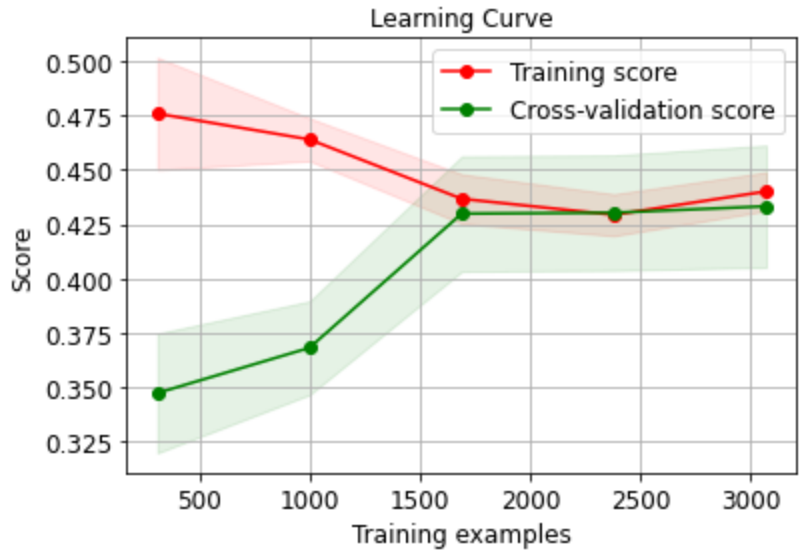
```
1 fun(svr)
```

```
---- SVR() ----
Taining Score:- 73.13828861118705
Mean Absolute Error 0.08908887394874306
Mean Squared Error 0.012111959920382892
Root Mean Squared Error 0.11005434984762252
Cross Validation Score -1.2246776424904704
R2 Score 0.669405019501851
Test Score 66.94050195018511
Model Peformance Cure
```



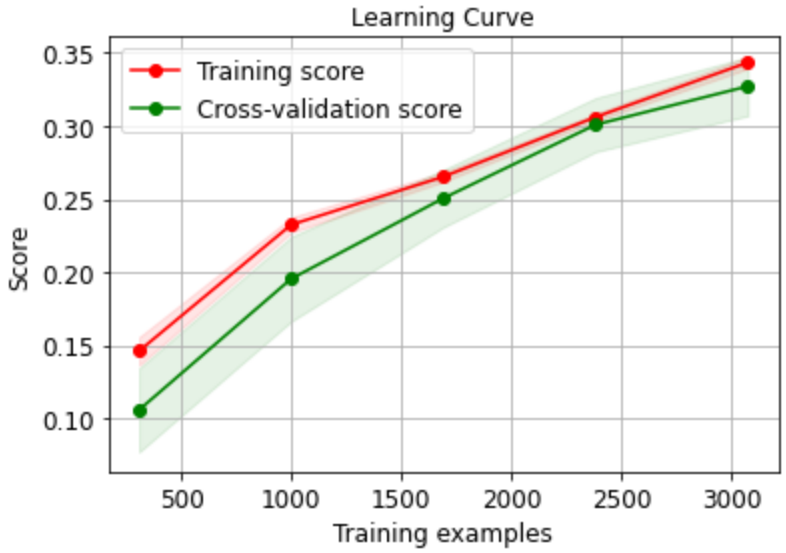
```
1 fun(svrl)
```

```
---- SVR(kernel='linear') ----
Taining Score:- 45.90714694922361
Mean Absolute Error 0.11329785701345703
Mean Squared Error 0.019570068519545986
Root Mean Squared Error 0.13989306101285362
Cross Validation Score 41.12471561668644
R2 Score 0.465836539825487
Test Score 46.583653982548704
Model Peformance Cure
```



```
1 fun(svrp)
```

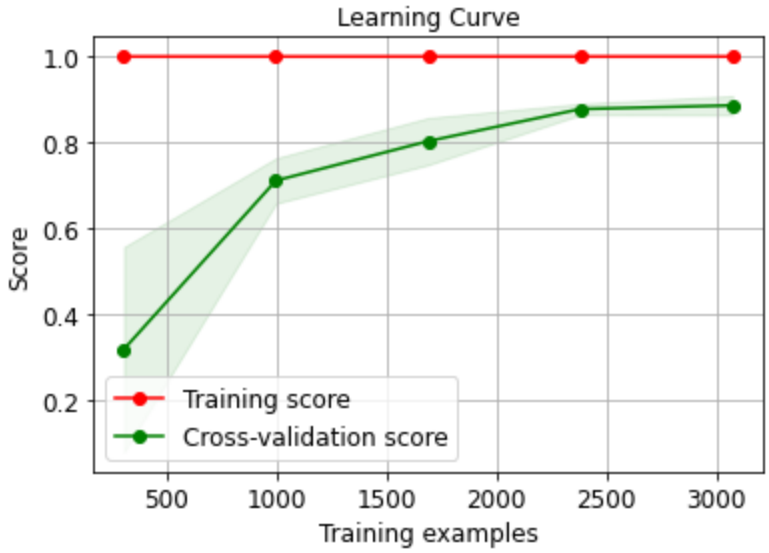
```
---- SVR(kernel='poly') ----
Taining Score:- 60.14804070690789
Mean Absolute Error 0.10505352865123023
Mean Squared Error 0.017185247121193462
Root Mean Squared Error 0.13109251359705276
Cross Validation Score -1.144240313811573
R2 Score 0.5309300497827918
Test Score 53.09300497827918
Model Peformance Cure
```



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.

```
1 fun(dtc)
```

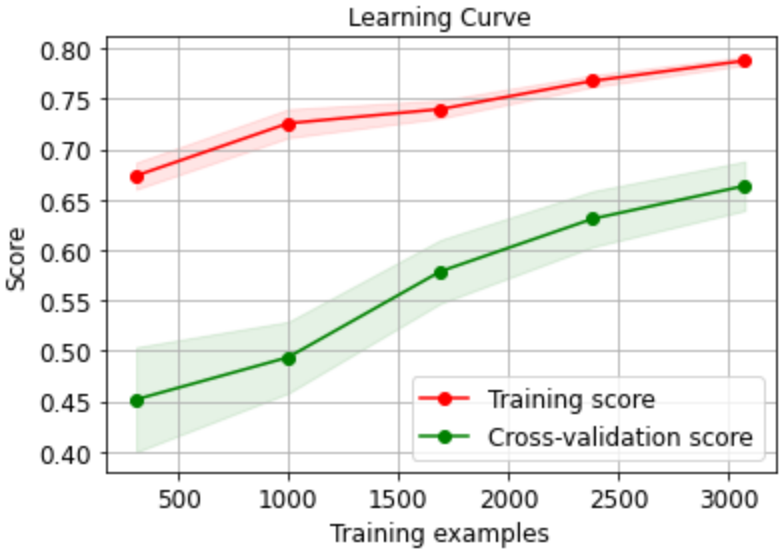
```
---- DecisionTreeRegressor() ----
Taining Score:- 99.99992423362168
Mean Absolute Error 0.03164478228437468
Mean Squared Error 0.004408086547579795
Root Mean Squared Error 0.06639342247225846
Cross Validation Score 87.5646226142729
R2 Score 0.8796816290830967
Test Score 87.96816290830967
Model Peformance Cure
```



KNN Regressor: KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same *neighborhood*. The size of the neighborhood needs to be set by the analyst or can be chosen using cross-validation (we will see this later) to select the size that minimizes the mean-squared error.

```
1 fun(knn)
```

```
---- KNeighborsRegressor() ----
Taining Score:- 77.63855928166599
Mean Absolute Error 0.08523320550575275
Mean Squared Error 0.012630246278323395
Root Mean Squared Error 0.11238436847855397
Cross Validation Score 45.52090826571932
R2 Score 0.6552584346780812
Test Score 65.52584346780812
Model Peformance Cure
```

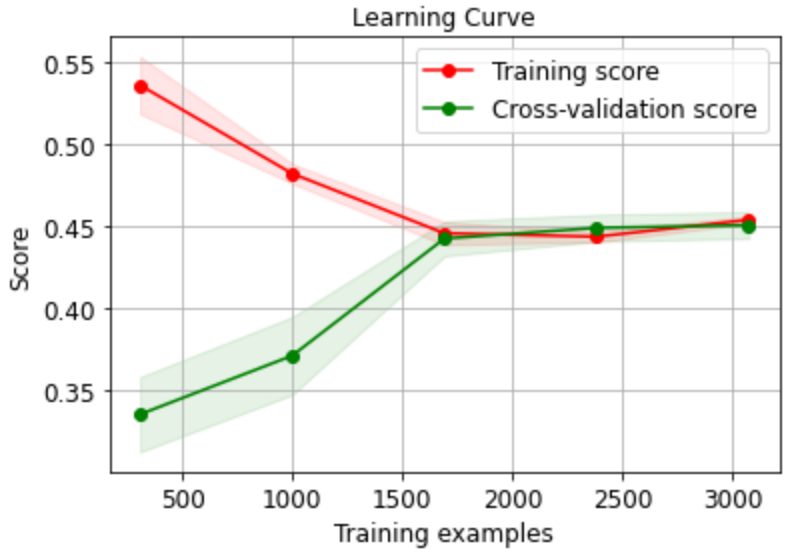


Lasso Regression: Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean. The lasso procedure encourages simple, sparse models (i.e., models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

Lasso Regression uses L1 regularization technique (will be discussed later in this article). It is used when we have more number of features because it automatically performs feature selection.

```
1 fun(ls)
```

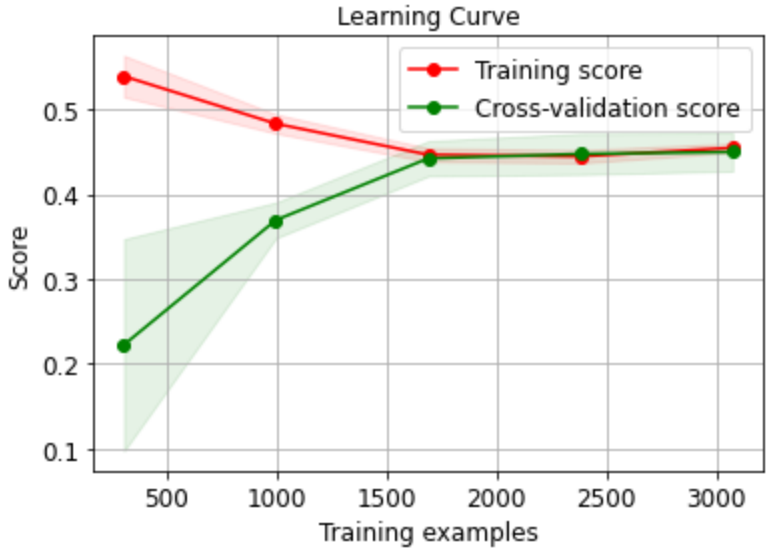
```
---- Lasso() ----
Taining Score:- 0.0
Mean Absolute Error 0.15598258732373504
Mean Squared Error 0.03668127029115349
Root Mean Squared Error 0.1915235502259539
Cross Validation Score 44.75563083280556
R2 Score -0.0012123484774431148
Test Score -0.12123484774431148
Model Peformance Cure
```



2) Ridge Regression: Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values.

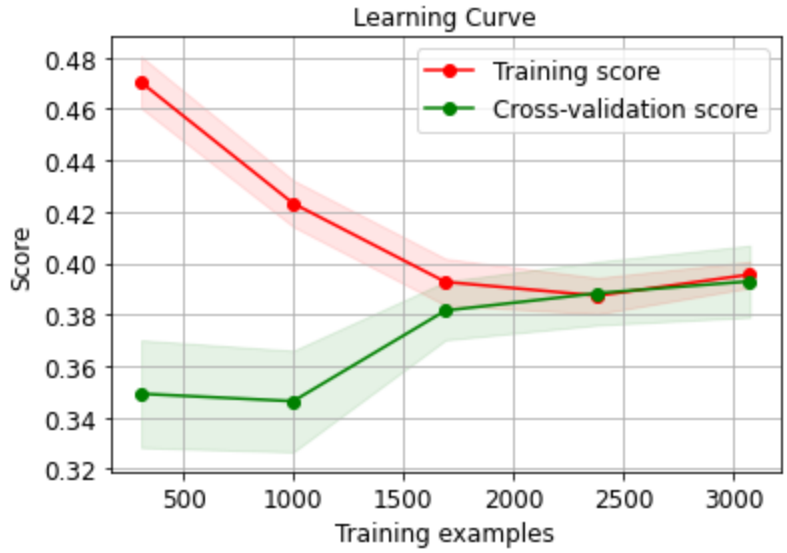
```
1 fun(rd)
```

```
---- Ridge() ----
Taining Score:- 46.131202051213926
Mean Absolute Error 0.11336500426629505
Mean Squared Error 0.019501407140922338
Root Mean Squared Error 0.13964743871952087
Cross Validation Score 44.99019068884049
R2 Score 0.4677106466815426
Test Score 46.77106466815426
Model Peformance Cure
```



```
1 fun(es)
```

```
---- ElasticNet() ----
Taining Score:- 0.0
Mean Absolute Error 0.15598258732373504
Mean Squared Error 0.03668127029115349
Root Mean Squared Error 0.1915235502259539
Cross Validation Score 32.59122413762034
R2 Score -0.0012123484774431148
Test Score -0.12123484774431148
Model Peformance Cure
```




```
1 from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor,BaggingRegressor
2 from sklearn.linear_model import SGDRegressor
```

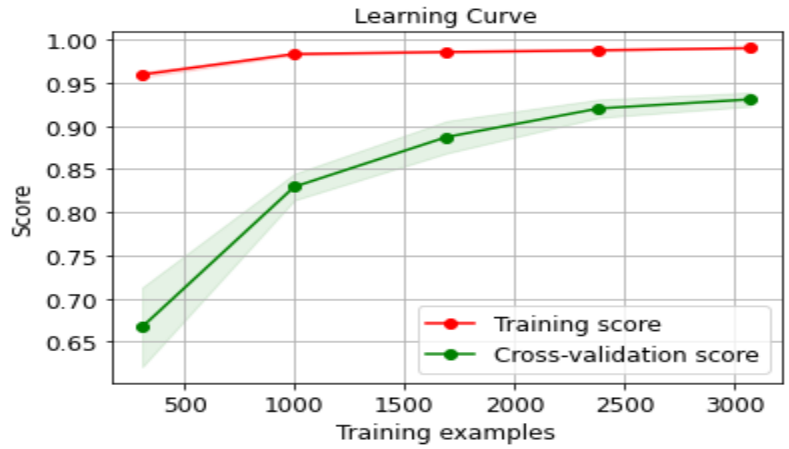
```
1 rf=RandomForestRegressor()
2 ad=AdaBoostRegressor()
3 gd=GradientBoostingRegressor()
4 sgd=SGDRegressor()
5 bg=BaggingRegressor()
```

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.

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.

```
1 fun(rf)
```

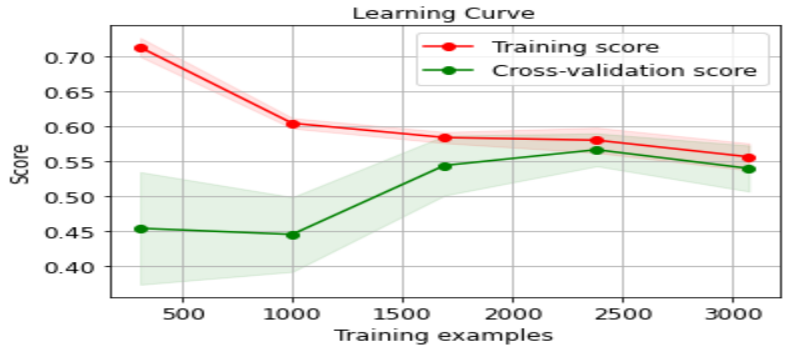
```
---- RandomForestRegressor() ----
Taining Score:- 99.01738206405936
Mean Absolute Error 0.030822128233702405
Mean Squared Error 0.0024303026874584284
Root Mean Squared Error 0.04929810024187979
Cross Validation Score 93.24665053384888
R2 Score 0.9336650818821799
Test Score 93.36650818821799
Model Peformance Cure
```



Ada Boost Regression: An AdaBoost regressor is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction. As such, subsequent regressors focus more on difficult cases.

```
1 fun(ad)
```

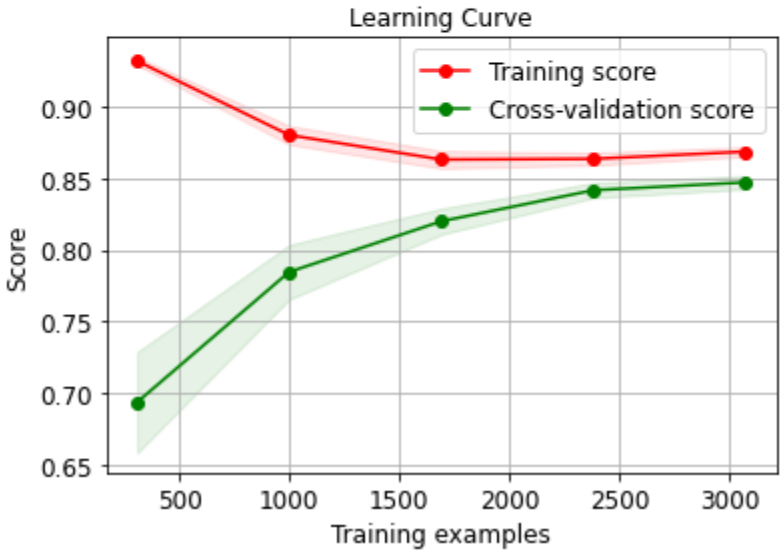
```
---- AdaBoostRegressor() ----
Taining Score:- 63.520301113171726
Mean Absolute Error 0.09604520626423219
Mean Squared Error 0.014153620824418553
Root Mean Squared Error 0.11896899102042748
Cross Validation Score 55.7559413571819
R2 Score 0.6136780478812114
Test Score 61.36780478812114
Model Peformance Cure
```



Gradient Boost Regressor: 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.

```
1 fun(gd)
```

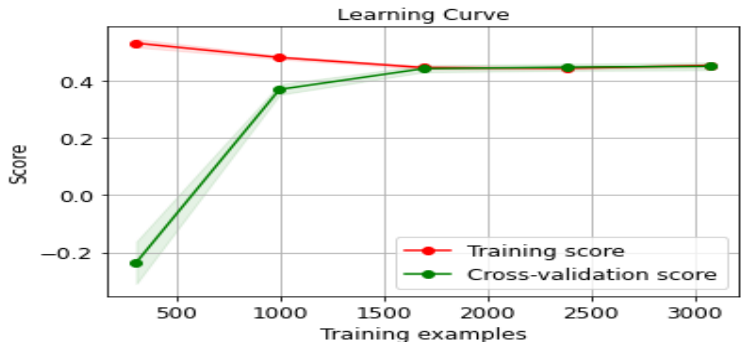
```
---- GradientBoostingRegressor() ----
Taining Score:- 87.97255432181281
Mean Absolute Error 0.05504449228006125
Mean Squared Error 0.0052059084928402525
Root Mean Squared Error 0.07215198190514417
Cross Validation Score 84.40297595249048
R2 Score 0.8579051426871578
Test Score 85.79051426871578
Model Peformance Cure
```



Stochastic Gradient Regressor: Stochastic Gradient Descent (SGD) regressor basically implements a plain SGD learning routine supporting various loss functions and penalties to fit linear regression models. Scikit-learn provides SGDRegressor module to implement SGD regression.

```
1 fun(sgd)
```

```
---- SGDRegressor() ----
Taining Score:- 46.030570057595035
Mean Absolute Error 0.11378723480085101
Mean Squared Error 0.01960252154652008
Root Mean Squared Error 0.14000900523366375
Cross Validation Score -1.0596677593669675e+28
R2 Score 0.464950737041286
Test Score 46.4950737041286
Model Peformance Cure
```

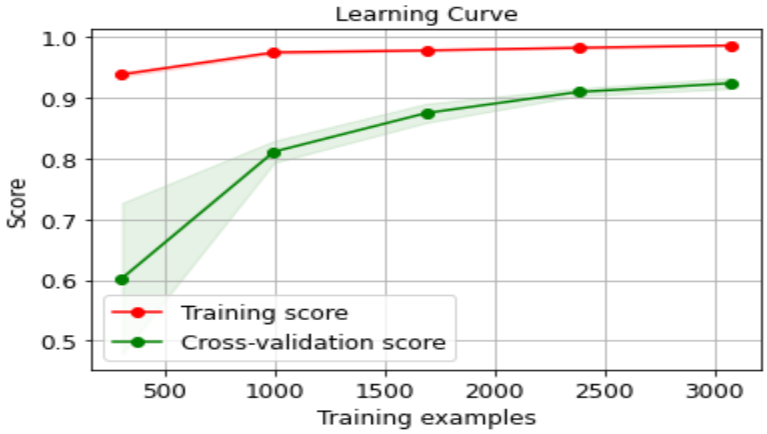


Baggage Regressor:

A Bagging regressor. A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. ... The base estimator to fit on random subsets of the dataset.

```
1 fun(bg)
```

```
---- BaggingRegressor() ----
Taining Score:- 98.45243446165122
Mean Absolute Error 0.03312819876478655
Mean Squared Error 0.0027311073678226274
Root Mean Squared Error 0.05225999777863206
Cross Validation Score 91.97647323644131
R2 Score 0.9254546421108757
Test Score 92.54546421108756
Model Peformance Cure
```



Now Let's Discuss About each variable in output:

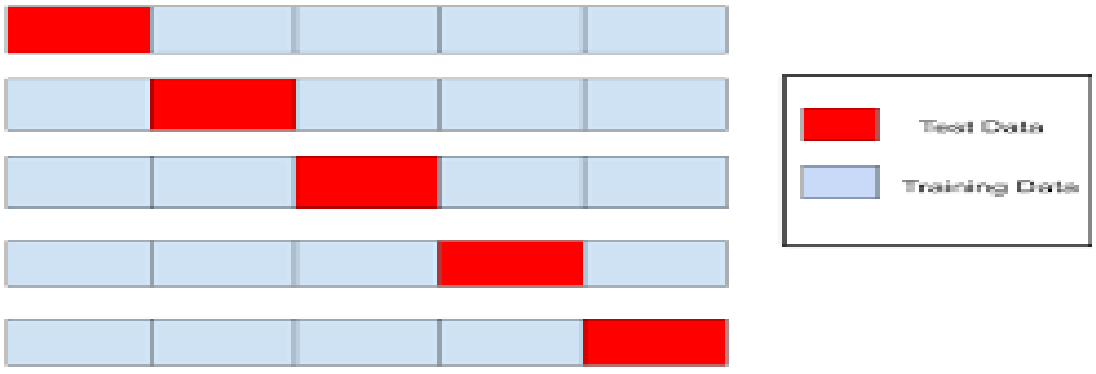
Mean Absolute Error: In statistics, mean absolute error (MAE) is **a measure of errors between paired observations expressing the same phenomenon**. ... This is known as a scale-dependent accuracy measure and therefore cannot be used to make comparisons between series using different scales.

Mean Squared Error: Mean Squared Error (MSD) of an estimator measures the average of error squares i.e., the average squared difference between the estimated values and true value. It is a risk function, corresponding to the expected value of the squared error loss. It is always non – negative and values close to zero are better. The MSE is the second moment of the error (about the origin) and thus incorporates both the variance of the estimator and its bias.

R2_score: Coefficient of determination also called as R^2 score is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.

Cross Validation: - This technique is used to check weather out data set is over fitting or under fitting. If model score is high and cv score is less it means model perform well in train dataset but did not perform well in unseen or test dataset. Feature selection is the best way to overcome the overfitting problem. There are 3 ways for the validation. KFold Cross validation score, Hold Out Methods and LOOCV.

KFold: - In this technique it will rotate the data into the k-fold times.



1st Iteration: 1-3 as Test and 4-9 Train

2nd Iteration:4-6 as Test and 1-3 & 7-9

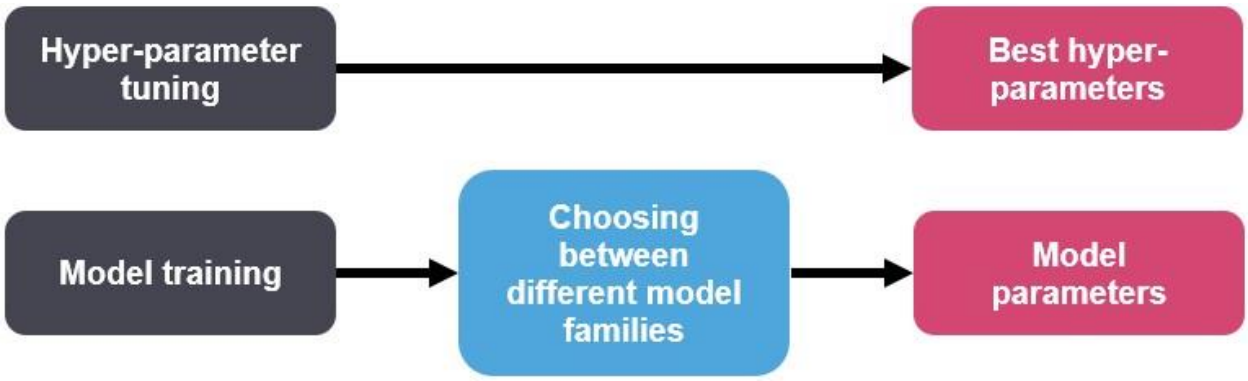
Train 3rd Iteration:7-9 as Test and 1-6 as Train

It means all the data (9 rows) go for training.

LOOCV: Leave one out cross validation It will take one row for test and remaining for training so each and every row go for test so its time-consuming processing

Hyper Tuning Method:

Hyper-parameters: Model parameters are learned from data and hyper-parameters are tuned to get the best fit. Searching for the best hyper-parameter can be tedious, hence search algorithms like grid search and random search are used.




```
1 from sklearn.model_selection import GridSearchCV
```

```
1 p1={'n_estimators':[100,150,250,500],'max_depth':[3,5,7,9,11],'min_samples_split':[1,2,3],'min_samples_leaf':[1,2],'max_feat
```

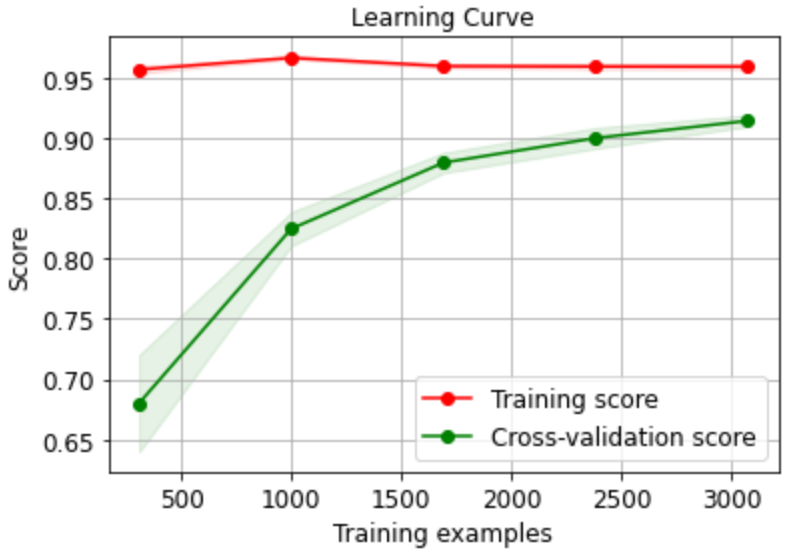
```
1 g1=GridSearchCV(rf,p1)
2 g1.fit(x,y)
3 print(g1.best_params_)
```

```
{'max_depth': 11, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}
```

```
1 rfr1=RandomForestRegressor(max_depth=11,max_features='auto',min_samples_leaf=1,n_estimators=500,min_samples_split=2)
```

```
1 fun(rfr1)
```

```
---- RandomForestRegressor(max_depth=11, n_estimators=500) ----
Taining Score:- 96.27687199186984
Mean Absolute Error 0.03798637979000832
Mean Squared Error 0.0031237547502536156
Root Mean Squared Error 0.05589056047539348
Cross Validation Score 90.96921017389457
R2 Score 0.9147373631080801
Test Score 91.47373631080801
Model Peformance Cure
```



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.

We used different regression methods to test the process of the model like Linear Regression, Decision tree Regression, Lasso Regression, Ridge Regression and Ada boosting Regression, Random Forest Regression, Gradient Regression, SGD Regression.

We get good score in Ridge Regressor got r2_score of 96.27% on training data, mean absolute error is 3.79%, Diff Between score and validation score also nice but not in 1st position but comparatively with all the variables and cross validation value also is high in Random Forest Regressor. the model performance is excellent.