USED CAR PRICE PREDICTION

A project report submitted to

DR. SUGANYA G

SCHOOL OF COMPUTER SCIENCE and ENGINEERING

is a partial fulfillment for the course

CSE4036 – MACHINE LEARNING

Vellore Institute of Technology

(Deemed to be University under section 3 of the UGC Act, 1956)

Name	**Reg. No.**	**Phone No.**
Atul Patel	20MIA1134	9179210782
Shashank Pandey	20MIA1147	9718630685
Piyali Saha	20MIA1066	6290612813
Chetan Srivastava	20MIA1122	6307696635

Abstract

New car prices in the industry are set by manufacturers and additional costs to the government in the form of taxes. Customers who purchase this new vehicle can therefore be confident that the amount they are investing is reasonable. However, sales of used cars increased worldwide as the prices of new cars skyrocketed and customers could no longer afford to buy them. Therefore, there is an urgent need for a he used car price prediction system that effectively determines the he value of a car based on various characteristics. This existing system involves a process in which sellers randomly set prices, and buyers know nothing about cars and their value in today's scenario. In fact, the seller also does not know the current value of the car or the price at which it should be sold. To solve this problem, we developed model, which is very effective. – A car price prediction has been a high interest research area, as it requires noticeable effort and knowledge of the field expert. The model predicts used cars reasonable prices based on multiple aspects, including vehicle mileage, year of manufacturing, fuel consumption, transmission, fuel type, and engine size. This model can benefit sellers, buyers, and car manufacturers in the used cars market. Regression algorithms are used because they give continuous values as output rather than categorized values. This makes it possible to predict the actual price of a car rather than its price range. A user interface has also been developed that captures input from any user and displays the car price according to the user's input.

Key Words

- Linear Regression
- Decision Tree Regressor
- Random forest Regressor
- Gradient boosting
- XG boost regressor
- Hyper-parameter tuning
- Cross- validation

Introduction

Due to the many factors that determine the price of used cars on the market, it is difficult to determine whether listing prices for used cars is a difficult task. The focus of this project is to develop this machine learning model that can accurately predict used car prices based on used car characteristics and make informed purchases. We implement and evaluate different learning methods on a dataset consisting of retail prices of various makes and models. Compare the performance of different machine learning algorithms such as Linear Regression, Decision Tree Regressor, Random-forest, Gradient boosting, XG boost regressor and choose the best. Depending on various parameters, the price of the car is determined. Car price prediction is somehow interesting and popular problem. Accurate car price prediction involves expert knowledge, because price usually depends on many distinctive features and factors. Typically, most significant ones are brand, year, power and mileage. The fuel type used in the car as well as fuel consumption per mile highly affect price of a car due to a frequent change in the price of a fuel. In this project, we applied different methods and techniques in order to achieve higher precision of the used car price prediction.

Background study

Predicting price of a used cars has been studied extensively in various researches. Listian discussed, in her paper written for Master thesis, that regression model that was built using Support Vector Machines (SVM) can predict the price of a car that has been leased with better precision than multivariate regression or some simple multiple regression. This is on the grounds that Support Vector Machine (SVM) is better in dealing with datasets with more dimensions and it is less prone to overfitting and underfitting. The weakness of this research is that a change of simple regression with more advanced SVM regression was not shown in basic indicators like mean, variance or standard deviation.

Another approach was given by Richardson in his thesis work. His theory was that car producers produce more durable cars. Richardson applied multiple regression analysis and demonstrated that hybrid cars retain their value for longer time than traditional cars. This has roots in environmental concerns about the climate and it gives higher fuel efficiency. They look at how supervised machine learning techniques can be used to estimate the price of second-hand cars in Mauritius in this study. The forecasts are based on historical data taken from daily publications. To make the predictions, various techniques such as linear regression analysis, were employed.

According to author Sameerchand, car price estimates on historical data gathered from daily newspapers. For estimating the price of cars, they employed supervised machine learning algorithms. Other methods that have been employed include multiple linear regression, and various decision tree algorithms. The best algorithm for prediction was identified after comparing all four algorithms. They had some issues comparing the algorithms, but they succeeded to do so.

Wu et al. in his study, exhibit automobile price prediction using a neural fuzzy knowledge-based system. They projected a model that has similar outcomes to the simple regression model by taking into account the following attributes: brand, year of manufacturing, and kind of engine. They have developed an expert system called ODAV (Optimal Distribution of Auction Automobiles) because there is a strong demand for car dealers to sell leased vehicles at the end of the lease year. This system gives insights into the best prices for vehicles, as well as the location where the best price can be gained. Regression model based on k-nearest neighbor machine learning algorithm was used to predict the price of a car. This system has a tendency to be exceptionally successful since more than two million vehicles were exchanged through it.

Noor and Jan build a model for car price prediction by using multiple linear regression. The dataset was created during the two-months period and included the following features: price, cubic capacity, exterior color, date when the ad was posted, number of ad views, power steering, mileage in kilometer, rims type, type of transmission, engine type, city, registered city, model, version, make and model year. After applying feature selection, the authors considered only engine type, price, model year and model as input features. With the given setup authors were able to achieve prediction accuracy of 98%. In the related work shown above, authors proposed prediction model based on the single machine learning algorithm.

Peerun et al. conducted study to assess the neural network's performance in predicting used automobile prices. However, especially on higher-priced cars, the estimated value is not very close to the real price. In forecasting the price of a used car, they found that support vector machine regression outperformed neural networks and linear regression by a little margin.

However, it is observed that single machine learning algorithm approach did not give remarkable prediction results and could be enhanced by assembling various machine learning methods in an ensemble to give more precision result.

<u>Idea and Proposed Methodology</u>

Ideology

Nearly 40% of India’s Population falls under the middle-class category. - Survey by Krishnan and Hatekar. Nowadays the ridesharing market is functioning on resold cars and in today's world customized cars have become the next new thing.

Automobile industry is one of the markets where used cars are bought and sold, and in today's market Predicting the resale value of used cars is an important aspect. With the ever-growing number of used car sales thanks to the booming ridesharing market, it is of paramount importance that the buyer/seller ‘s rupee is valued.

Methodology

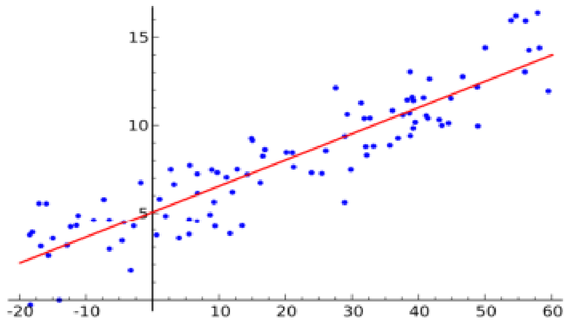
There are two primary phases in the system:

1. <u>Training phase:</u> The system is trained by using the data in the data set and fits a model (line/curve) based on the algorithm chosen accordingly.

2. <u>Testing phase</u>: the system is provided with the inputs and is tested for its working. The accuracy is checked. And therefore, the data that is used to train the model or test it, has to be appropriate. The system is designed to detect and predict price of used car and hence appropriate algorithms must be used to do the two different tasks. Before the algorithms are selected for further use, different algorithms were compared for its accuracy. The well-suited one for the task was chosen.

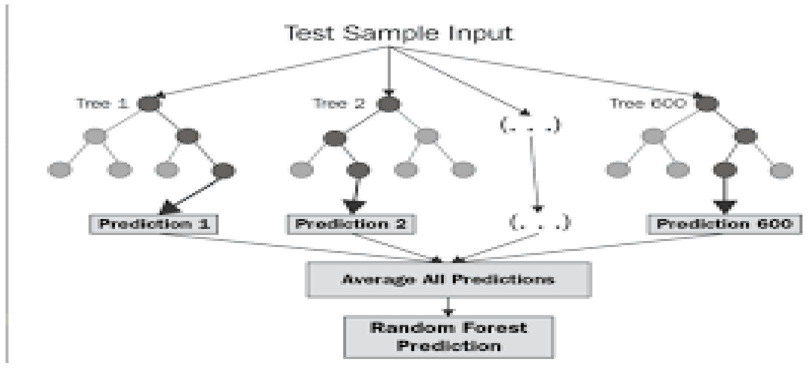
The main goal is to give users an accurate estimate of how much has to be paid for the given vehicle. The model may give the customer a record of possibilities for various automobiles based on the details of the automobile the customer wants. The system assists in providing the customer with sufficient data to help him to reach a conclusion. The used automobile market is expanding at an exponential rate, and vehicle vendors may profit from this by offering incorrect prices to capitalize on the demand. As a result, a system that can predict the price of a car based on its parameters while also taking into consideration the costs of competing vehicles is necessary. Our system fills in the gaps by providing buyers and sellers with an estimate of the car's value based on the best algorithm available for price prediction. This project aims to develop a good regression model to offer accurate prediction of car price. In order to do this, we need some previous data of used cars for which we use price and some other standard attributes. Car price is considered as the dependent variable while other attributes as the independent variables. Random Forest is an ensemble learning based regression model. It uses a model called decision tree, specifically as the name suggests, multiple decision trees to generate the ensemble model which collectively produces a prediction. The benefit of this model is that the trees are produced in parallel and are relatively uncorrelated, thus producing good results as each tree is not prone to individual errors of other trees. This uncorrelated behavior is partly ensured by the use of Bootstrap Aggregation or bagging providing the randomness required to produce robust and uncorrelated trees. This model was hence chosen to account for the large number of features in the dataset and compare a bagging technique with the following gradient boosting methods.

- LINEAR REGRESSION



Regression is a technique for predicting a dependent factor using independent factors. The approach is typically employed for predicting and calculating correlations between independent and dependent factors. The regression model establishes the relationship between independent factors and dependent factors i.e., linearly or exponentially. Linear regression is a sort of regression analysis in which there is only one independent variable and the independent(x) and dependent(y) variables can be bound in a linear relationship. In the graph above, the red line is called the best fit straight line. We aim to plot a line that best predicts the data-points based on the given data-points.

- RANDOM FOREST REGRESSION



It is a supervised learning algorithm that uses an ensemble learning method for classification and regression. Random forest is a bagging technique and not a boosting technique. The trees in random forests run in parallel, meaning is no interaction between these trees while building the trees. Random forest operates by constructing a multitude of decision trees at training time and outputting the class that's the mode of the classes (classification) or mean prediction (regression) of the individual trees.

It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier. It runs efficiently on large databases. It can handle thousands of input variables without variable deletion. It gives estimates of what variables are important in the classification. It generates an internal unbiased estimate of the generalization error as the forest building progresses.

It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

We will use the sklearn module for training our random forest regression model, specifically the RandomForestRegressor function using different parameters, few of those parameters are:

n_estimators — the number of decision trees you will be running in the model

criterion — this variable allows you to select the criterion (loss function) used to determine model outcomes. We can select from loss functions such as mean squared error (MSE) and mean absolute error (MAE). The default value is MSE.

max_depth — this sets the maximum possible depth of each tree

max_features — the maximum number of features the model will consider when determining a split

bootstrap — the default value for this is True, meaning the model follows bootstrapping principles

max_samples — This parameter assumes bootstrapping is set to True, if not, this parameter doesn't apply. In the case of True, this value sets the largest size of each sample for each tree.

- DECISION TREE REGRESSOR

  A Decision Tree is a supervised learning algorithm. It is a graphical representation of all the possible solutions. All the decisions were made based on some conditions. Decision Trees are a type of

Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

There are two main types of Decision Trees:

1. **Classification trees** (Yes/No types)

Here, the outcome is a variable like 'fit' or 'unfit'. Here the decision variable is **Categorical**.

2. **Regression trees** (Continuous data types)

Here the decision or the outcome variable is **Continuous**. **Working** . There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**

3. **Entropy:**

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

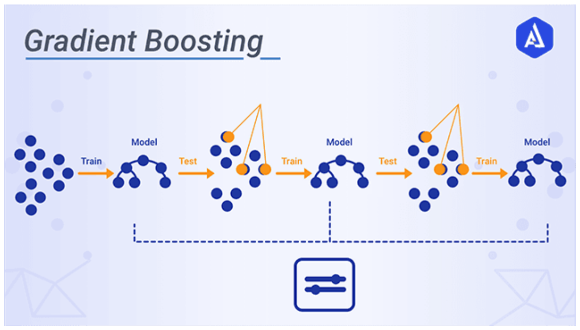
Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data. Intuitively, it tells us about the predictability of a certain event. Entropy to measure discriminatory power of an attribute for classification task. It defines the amount of randomness in attribute for classification task. Entropy is minimal means the attribute appears close to one class and has a good discriminatory power for classification

4. **Information Gain:**

$$IG(S, A) = H(S) - H(S, A)$$

Information gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent variables.

- GRADIENT BOOSTING



Gradient boosting is a technique used in creating models for prediction. The technique is mostly used in regression and classification procedures. Prediction models are often presented as decision trees for choosing the best prediction. Gradient boosting presents model building in stages, just like other boosting methods, while allowing the generalization and optimization of differentiable loss functions. The gradient boosting is also known as the statistical prediction model. It works quite similarly to other boosting methods even though it allows the generalization and optimization of the differential loss functions. One uses gradient boosting primarily in the procedures of regression and classification.

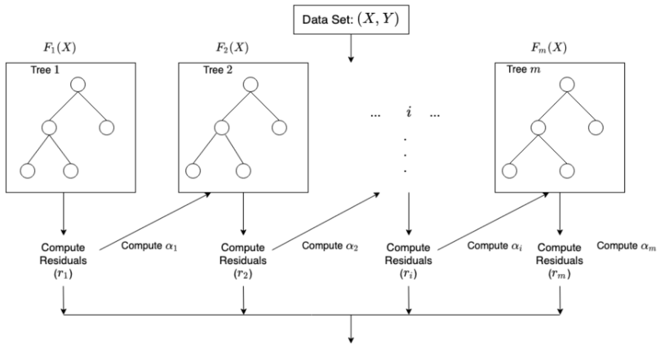
Gradient boosting creates prediction-based models in the form of a combination of weak prediction models. Weak hypotheses are parameters whose performance is slightly better than the randomly made choices. The gradient boosting method has witnessed many further developments to optimize the cost functions.

- XG BOOST

  XG Boost stands for extreme Gradient Boosting. XG Boost is an extension to gradient boosted decision trees (GBM) and specially designed to improve speed and performance.

XG Boost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XG Boost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.



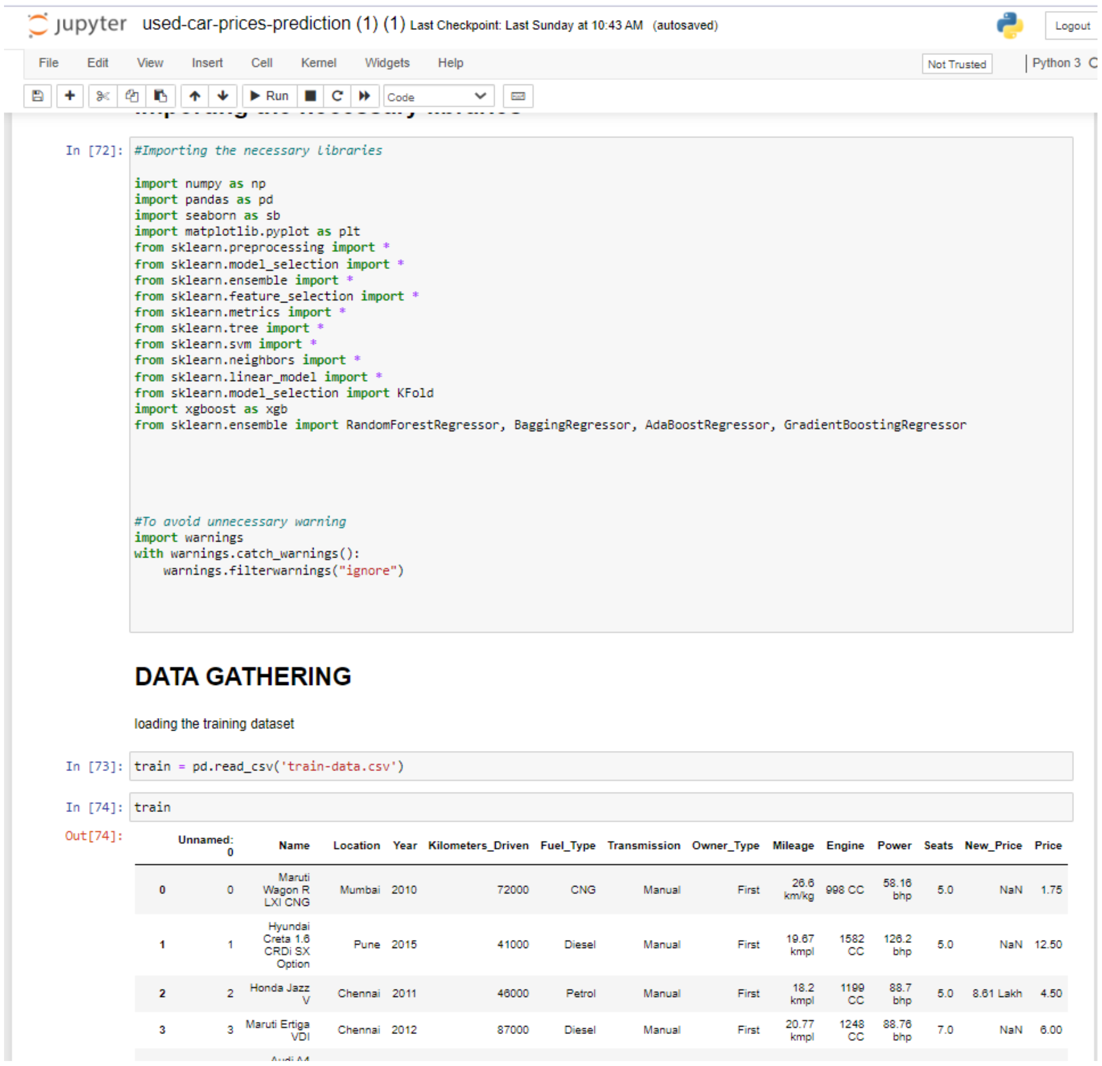
$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Requirements

- Hardware requirements
- Operating system- Windows 7,8,10
- Processor- dual core 2.4 GHz (i5 or i7 series Intel processor or equivalent AMD)
- RAM-4GB
- Software Requirements
- Python
- PyCharm
- PIP 2.7
- Jupyter Notebook
- Chrome

Implementation



jupyter used-car-prices-prediction (1) (1) Last Checkpoint: Last Sunday at 10:43 AM (autosaved)

```
In [72]: #Importing the necessary libraries

         import numpy as np
         import pandas as pd
         import seaborn as sb
         import matplotlib.pyplot as plt
         from sklearn.preprocessing import *
         from sklearn.model_selection import *
         from sklearn.ensemble import *
         from sklearn.feature_selection import *
         from sklearn.metrics import *
         from sklearn.tree import *
         from sklearn.svm import *
         from sklearn.neighbors import *
         from sklearn.linear_model import *
         from sklearn.model_selection import KFold
         import xgboost as xgb
         from sklearn.ensemble import RandomForestRegressor, BaggingRegressor, AdaBoostRegressor, GradientBoostingRegressor




         #To avoid unnecessary warning
         import warnings
         with warnings.catch_warnings():
             warnings.filterwarnings("ignore")
```

DATA GATHERING

loading the training dataset

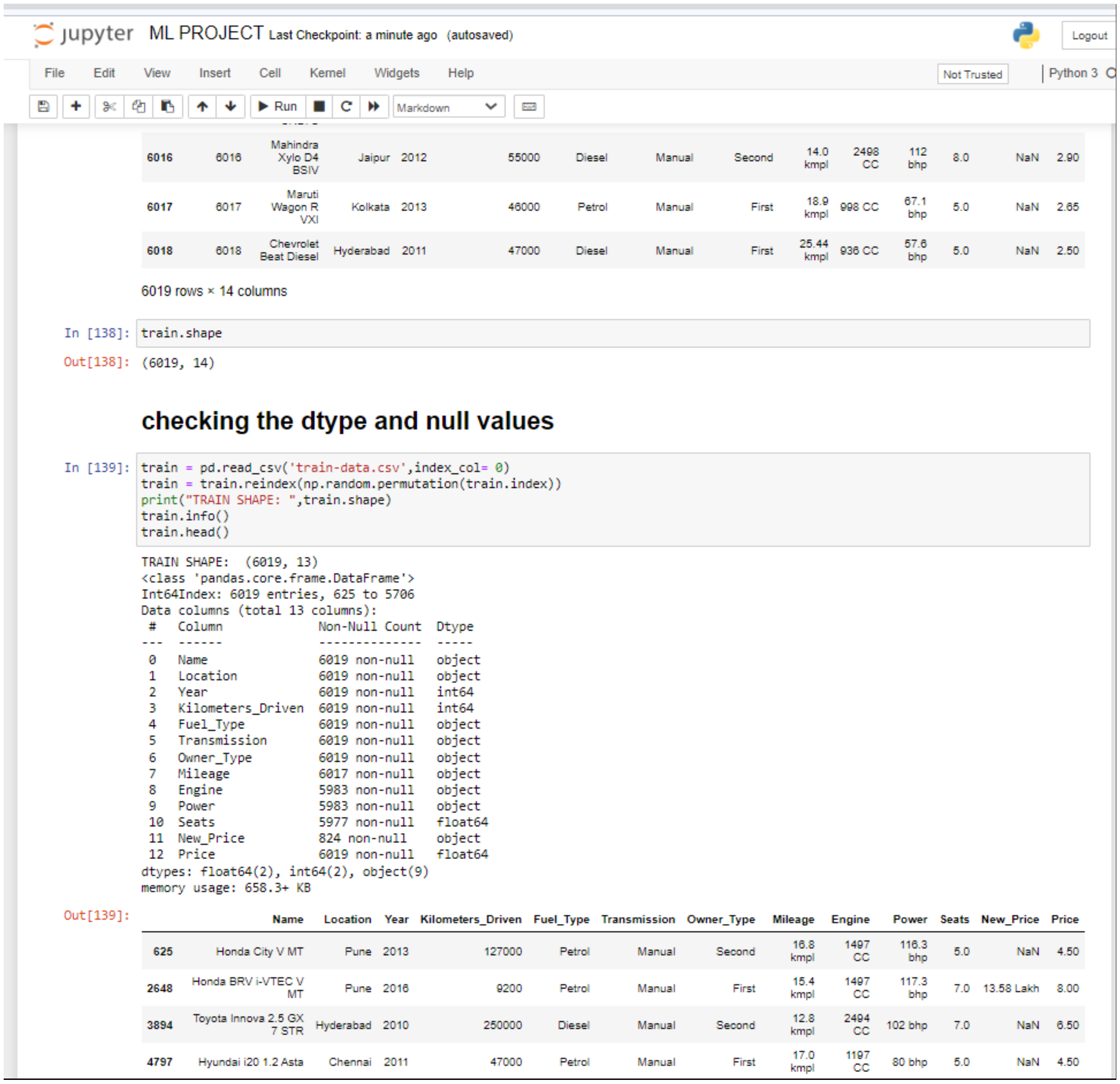
```
In [73]: train = pd.read_csv('train-data.csv')
```

```
In [74]: train
```

Out[74]:

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00

- Here we have imported all the libraries and loaded the dataset here we can see the dataset


jupyter ML PROJECT Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

6016	6016	Mahindra Xylo D4 BSIV	Jaipur	2012	55000	Diesel	Manual	Second	14.0 kmpl	2498 CC	112 bhp	8.0	NaN	2.90
6017	6017	Maruti Wagon R VXI	Kolkata	2013	46000	Petrol	Manual	First	18.9 kmpl	998 CC	67.1 bhp	5.0	NaN	2.65
6018	6018	Chevrolet Beat Diesel	Hyderabad	2011	47000	Diesel	Manual	First	25.44 kmpl	936 CC	57.6 bhp	5.0	NaN	2.50

6019 rows × 14 columns

```
In [138]: train.shape

Out[138]: (6019, 14)
```

checking the dtype and null values

```
In [139]: train = pd.read_csv('train-data.csv',index_col= 0)
          train = train.reindex(np.random.permutation(train.index))
          print("TRAIN SHAPE: ",train.shape)
          train.info()
          train.head()
```

```
TRAIN SHAPE:  (6019, 13)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 625 to 5706
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               6019 non-null   object
 1   Location           6019 non-null   object
 2   Year               6019 non-null   int64
 3   Kilometers_Driven  6019 non-null   int64
 4   Fuel_Type          6019 non-null   object
 5   Transmission       6019 non-null   object
 6   Owner_Type         6019 non-null   object
 7   Mileage            6017 non-null   object
 8   Engine             5983 non-null   object
 9   Power              5983 non-null   object
 10  Seats              5977 non-null   float64
 11  New_Price          824 non-null    object
 12  Price              6019 non-null   float64
dtypes: float64(2), int64(2), object(9)
memory usage: 658.3+ KB
```

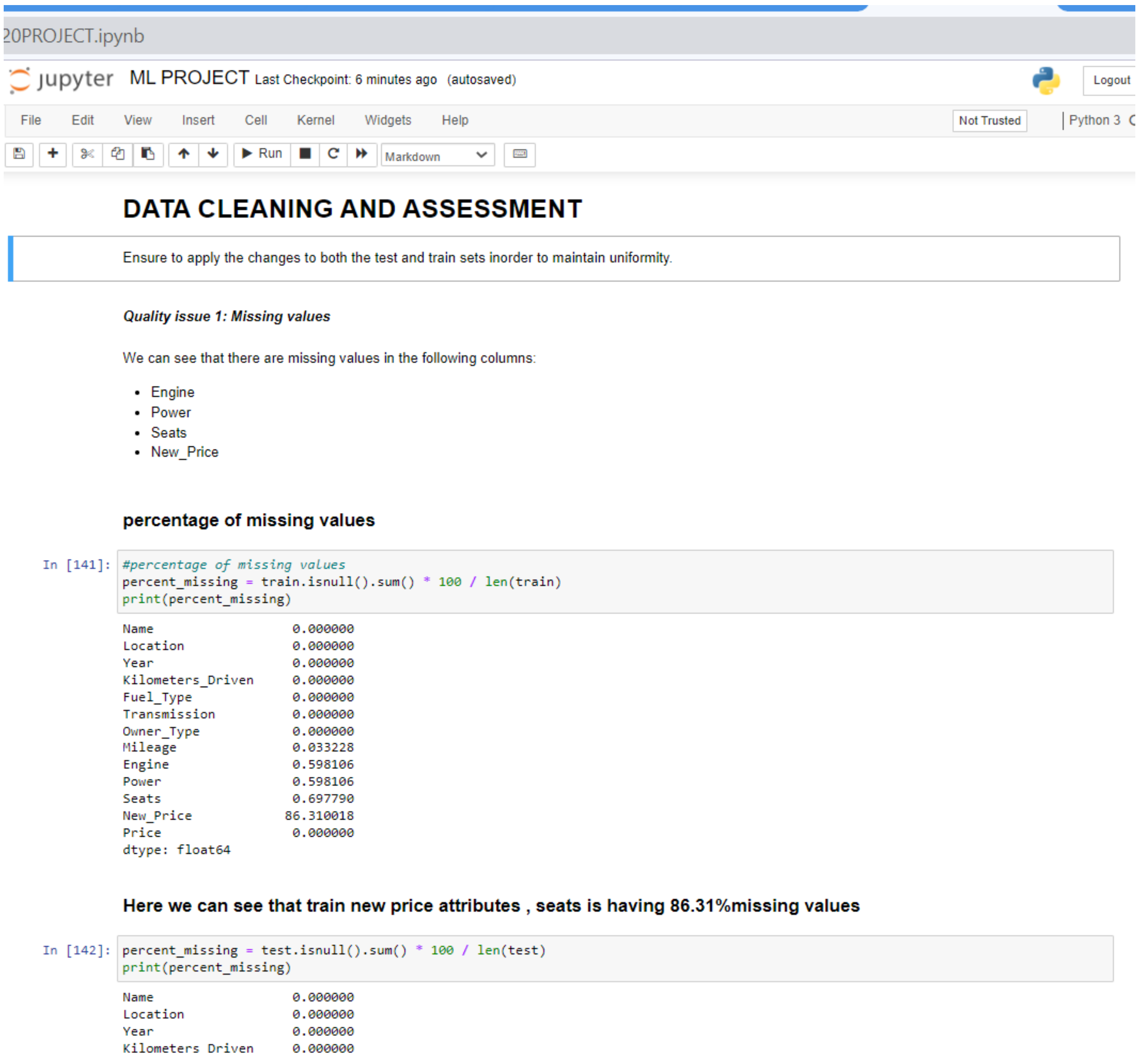
Out[139]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
625	Honda City V MT	Pune	2013	127000	Petrol	Manual	Second	16.8 kmpl	1497 CC	116.3 bhp	5.0	NaN	4.50
2648	Honda BRV i-VTEC V MT	Pune	2016	9200	Petrol	Manual	First	15.4 kmpl	1497 CC	117.3 bhp	7.0	13.58 Lakh	8.00
3894	Toyota Innova 2.5 GX 7 STR	Hyderabad	2010	250000	Diesel	Manual	Second	12.8 kmpl	2494 CC	102 bhp	7.0	NaN	6.50
4797	Hyundai i20 1.2 Asta	Chennai	2011	47000	Petrol	Manual	First	17.0 kmpl	1197 CC	80 bhp	5.0	NaN	4.50

- We can observe that shape of our dataset is 6019 rows and 14 columns of which 13columnn are independent and 1 which is a selling price is dependent or target column
- our dataset contain column such as location, year, kilometers driven, fuel type, Transmission , owner

Type, mileage, engine, power, seats, new_price, price .

- In our dataset year , kilometere driven , engine , mileage 4 coloumn are numerical 8 are categorical .
- In our dataset 2 values are float and 2 of them are integers and 9 of them are object


20PROJECT.ipynb
jupyter ML PROJECT Last Checkpoint: 6 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

DATA CLEANING AND ASSESSMENT

Ensure to apply the changes to both the test and train sets inorder to maintain uniformity.

Quality issue 1: Missing values

We can see that there are missing values in the following columns:

- Engine
- Power
- Seats
- New_Price

percentage of missing values

In [141]:
```
#percentage of missing values
percent_missing = train.isnull().sum() * 100 / len(train)
print(percent_missing)
```

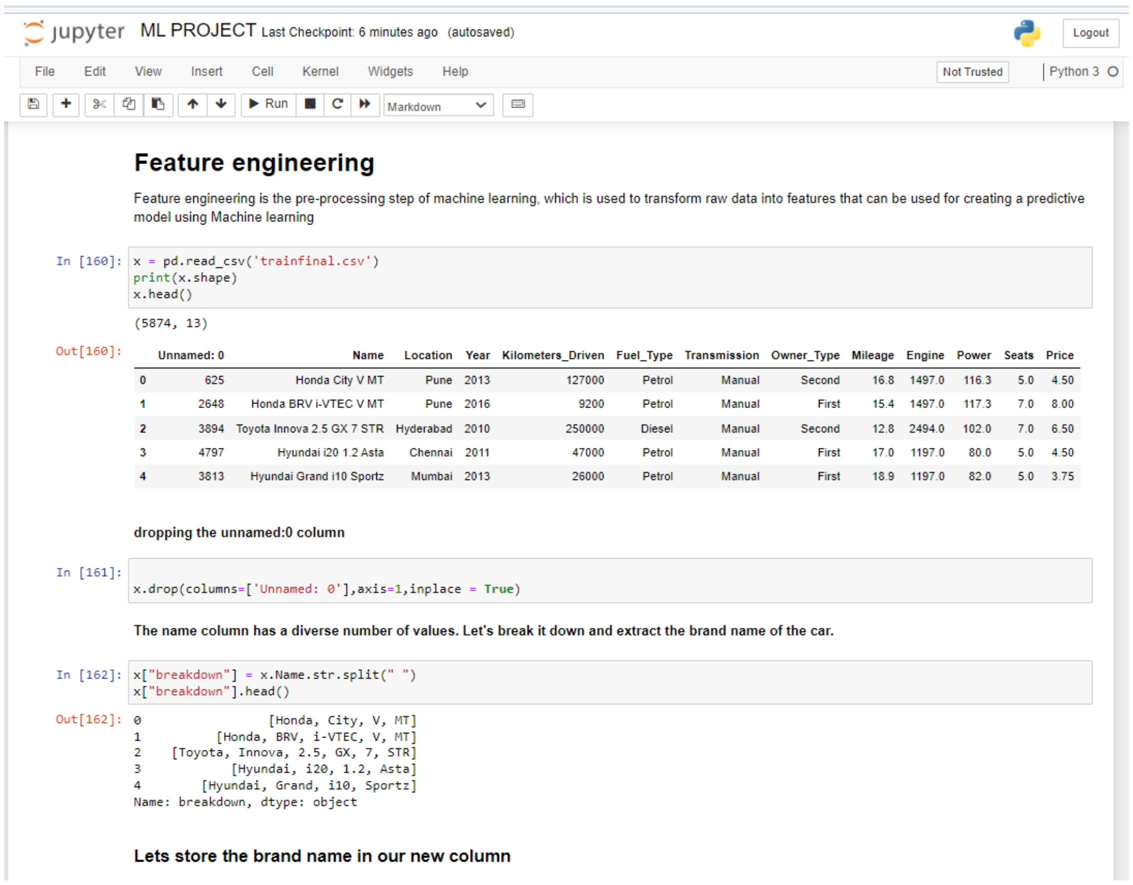
```
Name                  0.000000
Location              0.000000
Year                  0.000000
Kilometers_Driven     0.000000
Fuel_Type             0.000000
Transmission          0.000000
Owner_Type            0.000000
Mileage               0.033228
Engine                0.598106
Power                 0.598106
Seats                 0.697790
New_Price            86.310018
Price                 0.000000
dtype: float64
```

Here we can see that train new price attributes , seats is having 86.31%missing values

In [142]:
```
percent_missing = test.isnull().sum() * 100 / len(test)
print(percent_missing)
```

```
Name                  0.000000
Location              0.000000
Year                  0.000000
Kilometers_Driven     0.000000
```

- Here we have cleaned the data, have found the missing values, we can observe the new price attribute contains 86% missing values hence we have dropped the column.
- Now we can see the the missing % is now 0.


jupyter ML PROJECT Last Checkpoint: 6 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

Feature engineering

Feature engineering is the pre-processing step of machine learning, which is used to transform raw data into features that can be used for creating a predictive model using Machine learning

In [160]:
```
x = pd.read_csv('trainfinal.csv')
print(x.shape)
x.head()
```

```
(5874, 13)
```

Out[160]:

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price
0	625	Honda City V MT	Pune	2013	127000	Petrol	Manual	Second	16.8	1497.0	116.3	5.0	4.50
1	2648	Honda BRV i-VTEC V MT	Pune	2016	9200	Petrol	Manual	First	15.4	1497.0	117.3	7.0	8.00
2	3894	Toyota Innova 2.5 GX 7 STR	Hyderabad	2010	250000	Diesel	Manual	Second	12.8	2494.0	102.0	7.0	6.50
3	4797	Hyundai i20 1.2 Asta	Chennai	2011	47000	Petrol	Manual	First	17.0	1197.0	80.0	5.0	4.50
4	3813	Hyundai Grand i10 Sportz	Mumbai	2013	26000	Petrol	Manual	First	18.9	1197.0	82.0	5.0	3.75

dropping the unnamed:0 column

In [161]:
```
x.drop(columns=['Unnamed: 0'],axis=1,inplace = True)
```

The name column has a diverse number of values. Let's break it down and extract the brand name of the car.

In [162]:
```
x["breakdown"] = x.Name.str.split(" ")
x["breakdown"].head()
```

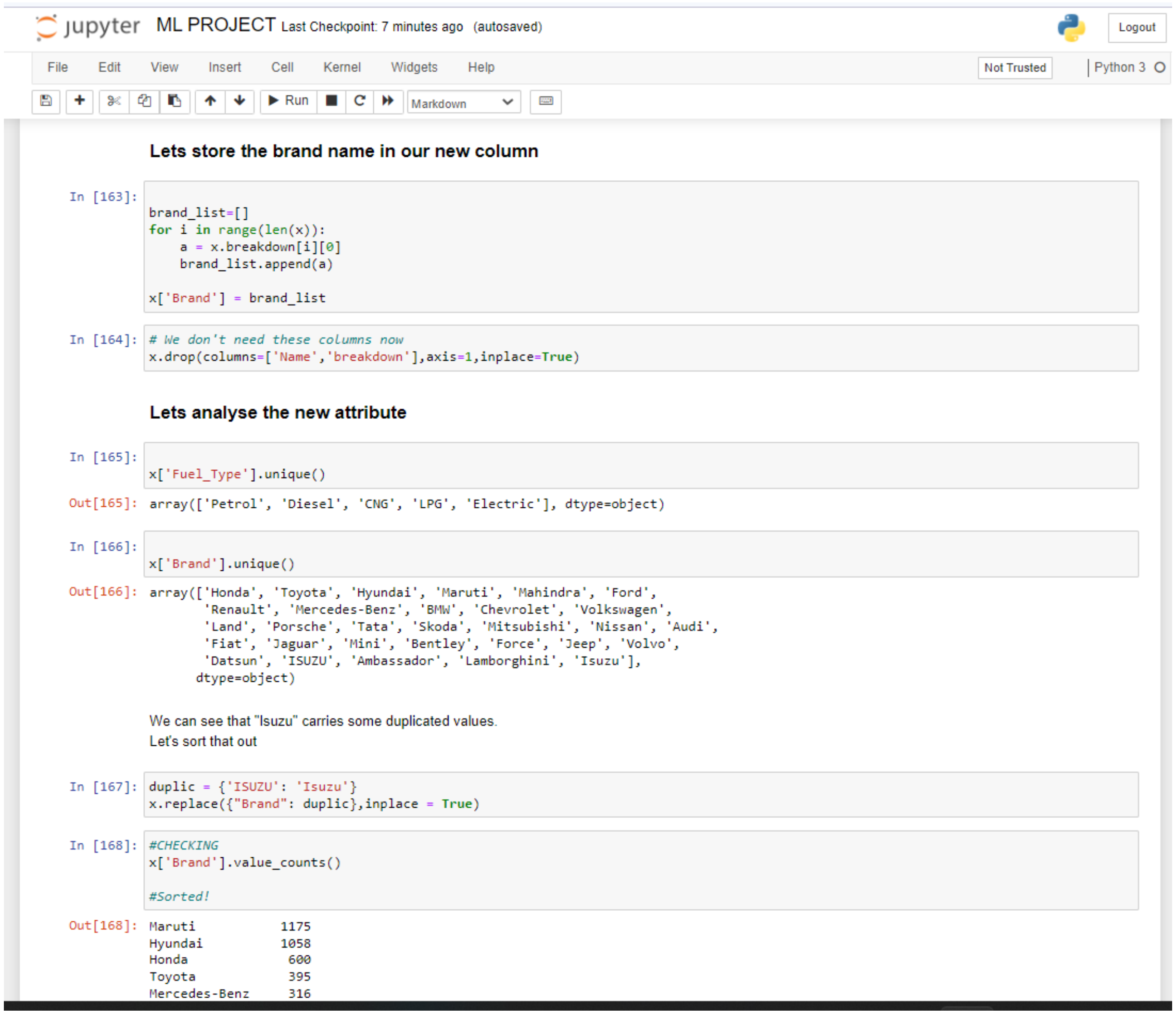
Out[162]:
```
0                  [Honda, City, V, MT]
1           [Honda, BRV, i-VTEC, V, MT]
2    [Toyota, Innova, 2.5, GX, 7, STR]
3             [Hyundai, i20, 1.2, Asta]
4         [Hyundai, Grand, i10, Sportz]
Name: breakdown, dtype: object
```

Lets store the brand name in our new column

- Here we have done the feature engineering with the preprocessed dataset and made the dataset fit for the modeling
- Here we have a breakdown of the name attribute in the comapany name and brand name

- And also analyse the new attributes such as brand and , fuel type etc

fu



jupyter ML PROJECT Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

Lets store the brand name in our new column

```
In [163]: brand_list=[]
          for i in range(len(x)):
              a = x.breakdown[i][0]
              brand_list.append(a)

          x['Brand'] = brand_list
```

```
In [164]: # We don't need these columns now
          x.drop(columns=['Name','breakdown'],axis=1,inplace=True)
```

Lets analyse the new attribute

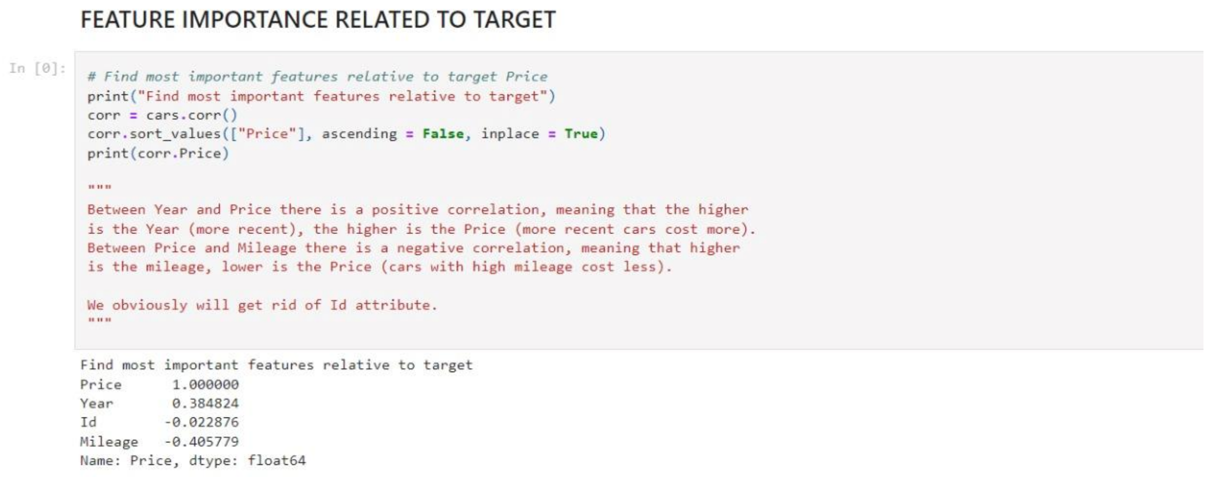
```
In [165]: x['Fuel_Type'].unique()

Out[165]: array(['Petrol', 'Diesel', 'CNG', 'LPG', 'Electric'], dtype=object)
```

```
In [166]: x['Brand'].unique()

Out[166]: array(['Honda', 'Toyota', 'Hyundai', 'Maruti', 'Mahindra', 'Ford',
                 'Renault', 'Mercedes-Benz', 'BMW', 'Chevrolet', 'Volkswagen',
                 'Land', 'Porsche', 'Tata', 'Skoda', 'Mitsubishi', 'Nissan', 'Audi',
                 'Fiat', 'Jaguar', 'Mini', 'Bentley', 'Force', 'Jeep', 'Volvo',
                 'Datsun', 'ISUZU', 'Ambassador', 'Lamborghini', 'Isuzu'],
                dtype=object)
```

We can see that "Isuzu" carries some duplicated values.
Let's sort that out

```
In [167]: duplic = {'ISUZU': 'Isuzu'}
          x.replace({"Brand": duplic},inplace = True)
```

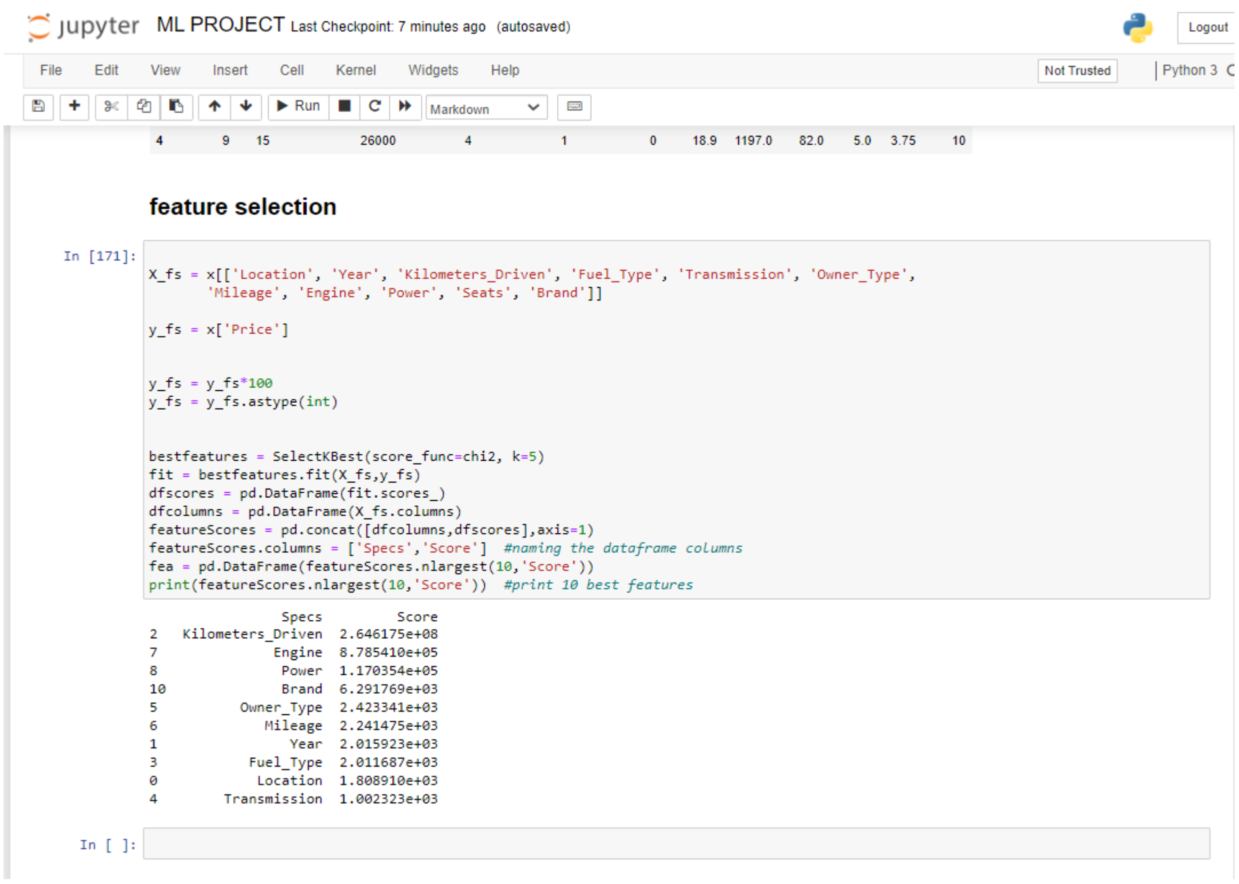
```
In [168]: #CHECKING
          x['Brand'].value_counts()

          #Sorted!

Out[168]: Maruti           1175
          Hyundai          1058
          Honda             600
          Toyota            395
          Mercedes-Benz     316
```


FEATURE IMPORTANCE RELATED TO TARGET

```
In [0]:
# Find most important features relative to target Price
print("Find most important features relative to target")
corr = cars.corr()
corr.sort_values(["Price"], ascending = False, inplace = True)
print(corr.Price)

"""
Between Year and Price there is a positive correlation, meaning that the higher
is the Year (more recent), the higher is the Price (more recent cars cost more).
Between Price and Mileage there is a negative correlation, meaning that higher
is the mileage, lower is the Price (cars with high mileage cost less).

We obviously will get rid of Id attribute.
"""
```

```
Find most important features relative to target
Price      1.000000
Year       0.384824
Id        -0.022876
Mileage   -0.405779
Name: Price, dtype: float64
```

- Here we have to find the correlation between the independent and dependent variable
- Also we have done feature selection, like what is the features which are mainly important for our prediction model
- We can observe in the below feature selection that transmission is the highest priority factor followed by kilometer driven than fuel type.


jupyter ML PROJECT Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

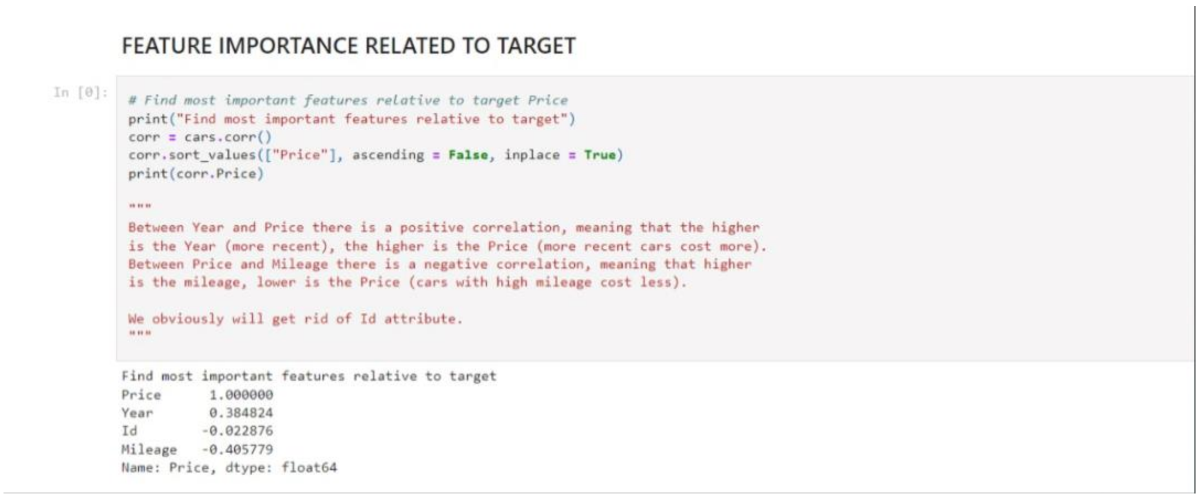
```
4    9   15          26000        4           1           0     18.9   1197.0    82.0    5.0   3.75     10
```

feature selection

In [171]:
```
X_fs = x[['Location', 'Year', 'Kilometers_Driven', 'Fuel_Type', 'Transmission', 'Owner_Type',
       'Mileage', 'Engine', 'Power', 'Seats', 'Brand']]

y_fs = x['Price']


y_fs = y_fs*100
y_fs = y_fs.astype(int)


bestfeatures = SelectKBest(score_func=chi2, k=5)
fit = bestfeatures.fit(X_fs,y_fs)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X_fs.columns)
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
fea = pd.DataFrame(featureScores.nlargest(10,'Score'))
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
                Specs         Score
2   Kilometers_Driven  2.646175e+08
7              Engine  8.785410e+05
8               Power  1.170354e+05
10              Brand  6.291769e+03
5          Owner_Type  2.423341e+03
6             Mileage  2.241475e+03
1                Year  2.015923e+03
3           Fuel_Type  2.011687e+03
0            Location  1.808910e+03
4        Transmission  1.002323e+03
```

In [ ]:



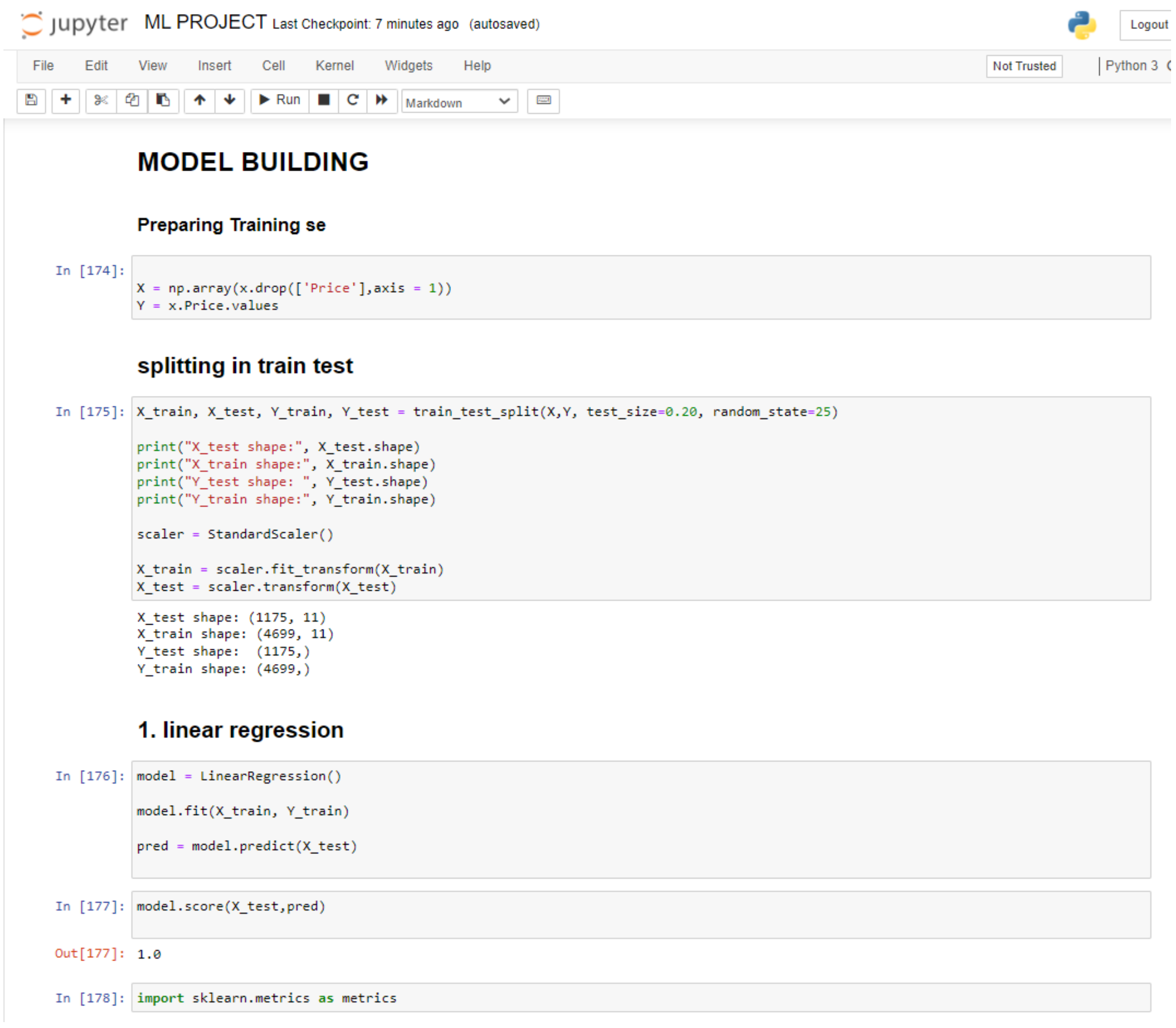
FEATURE IMPORTANCE RELATED TO TARGET

In [0]:
```
# Find most important features relative to target Price
print("Find most important features relative to target")
corr = cars.corr()
corr.sort_values(["Price"], ascending = False, inplace = True)
print(corr.Price)

"""
Between Year and Price there is a positive correlation, meaning that the higher
is the Year (more recent), the higher is the Price (more recent cars cost more).
Between Price and Mileage there is a negative correlation, meaning that higher
is the mileage, lower is the Price (cars with high mileage cost less).

We obviously will get rid of Id attribute.
"""
```

```
Find most important features relative to target
Price       1.000000
Year        0.384824
Id         -0.022876
Mileage    -0.405779
Name: Price, dtype: float64
```

- Here we have come to the model building part where we have split the data set into train and test with the ratio of 80:20.

- Then we have fitted the data by scaler transform
- Here we have applied various models which are as follows:-
  1. Linear regression & their metrics
  2. Decision
  3. Random
  4. gradient boosting
  5. xg boost


jupyter ML PROJECT Last Checkpoint: 7 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown

MODEL BUILDING

Preparing Training se

In [174]:
```
X = np.array(x.drop(['Price'],axis = 1))
Y = x.Price.values
```

splitting in train test

In [175]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.20, random_state=25)

print("X_test shape:", X_test.shape)
print("X_train shape:", X_train.shape)
print("Y_test shape: ", Y_test.shape)
print("Y_train shape:", Y_train.shape)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_test shape: (1175, 11)
X_train shape: (4699, 11)
Y_test shape:  (1175,)
Y_train shape: (4699,)
```

1. linear regression

In [176]:
```
model = LinearRegression()

model.fit(X_train, Y_train)

pred = model.predict(X_test)
```

In [177]:
```
model.score(X_test,pred)
```

Out[177]:
```
1.0
```

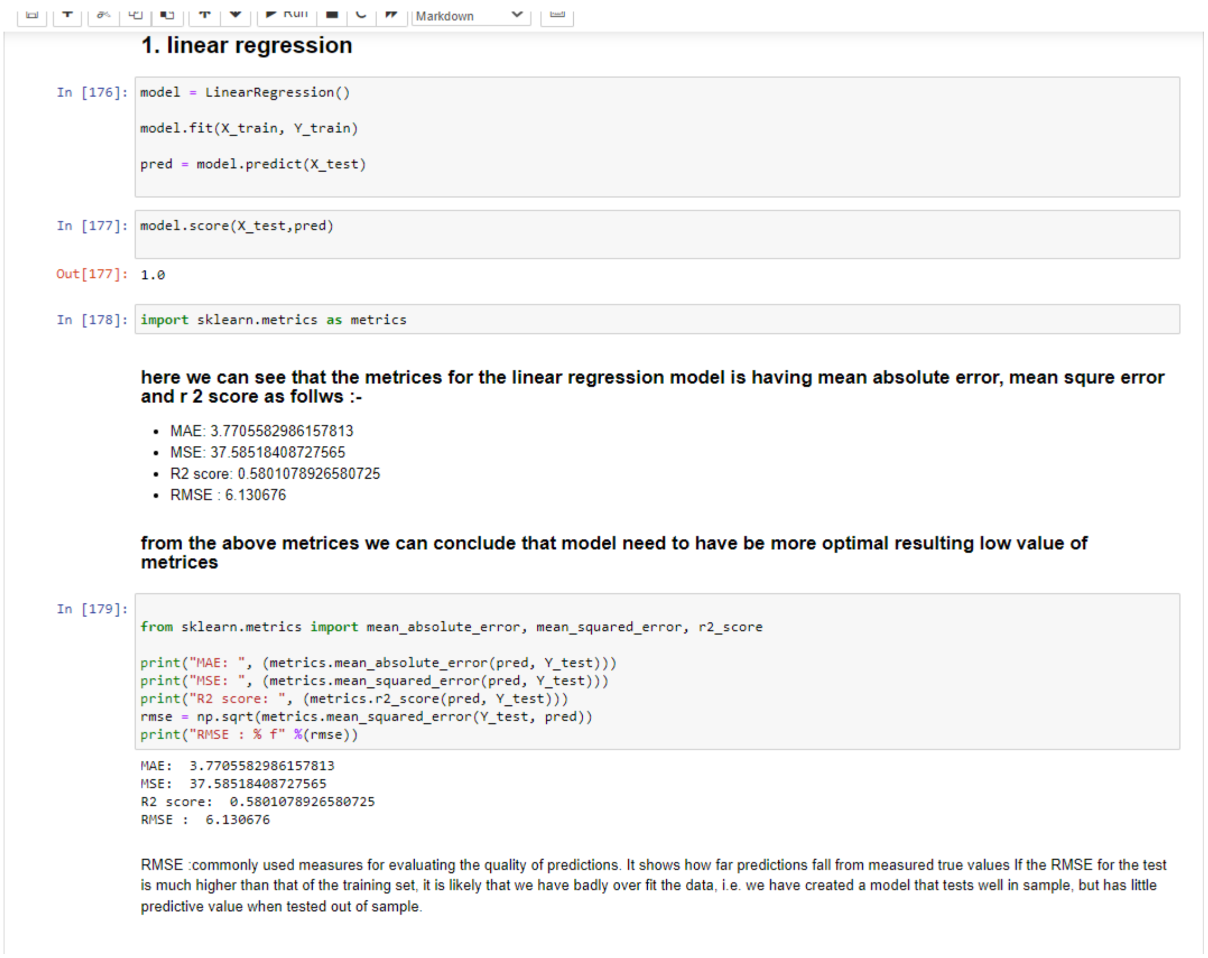
In [178]:
```
import sklearn.metrics as metrics
```

LINEAR REGRESSION:-

Here we can see the model got fitted and we have metrices such as

1. MAE SCORE:- 3.77
2. MSE SCORE:- 37.58
3. R2 SCORE :- 0.58
4. RMSE:- 6.130

   Here we can infer that the built model has produced the above metrics and this score such as MAE, MSE, and RMSE should be reduced ie on the other hand r2 score should be increased .



1. linear regression

```
In [176]: model = LinearRegression()

          model.fit(X_train, Y_train)

          pred = model.predict(X_test)

In [177]: model.score(X_test,pred)

Out[177]: 1.0

In [178]: import sklearn.metrics as metrics
```

here we can see that the metrices for the linear regression model is having mean absolute error, mean squre error and r 2 score as follws :-

- MAE: 3.7705582986157813
- MSE: 37.58518408727565
- R2 score: 0.5801078926580725
- RMSE : 6.130676

from the above metrices we can conclude that model need to have be more optimal resulting low value of metrices

```
In [179]:
          from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

          print("MAE: ", (metrics.mean_absolute_error(pred, Y_test)))
          print("MSE: ", (metrics.mean_squared_error(pred, Y_test)))
          print("R2 score: ", (metrics.r2_score(pred, Y_test)))
          rmse = np.sqrt(metrics.mean_squared_error(Y_test, pred))
          print("RMSE : % f" %(rmse))

          MAE:  3.7705582986157813
          MSE:  37.58518408727565
          R2 score:  0.5801078926580725
          RMSE :  6.130676
```

RMSE :commonly used measures for evaluating the quality of predictions. It shows how far predictions fall from measured true values If the RMSE for the test is much higher than that of the training set, it is likely that we have badly over fit the data, i.e. we have created a model that tests well in sample, but has little predictive value when tested out of sample.

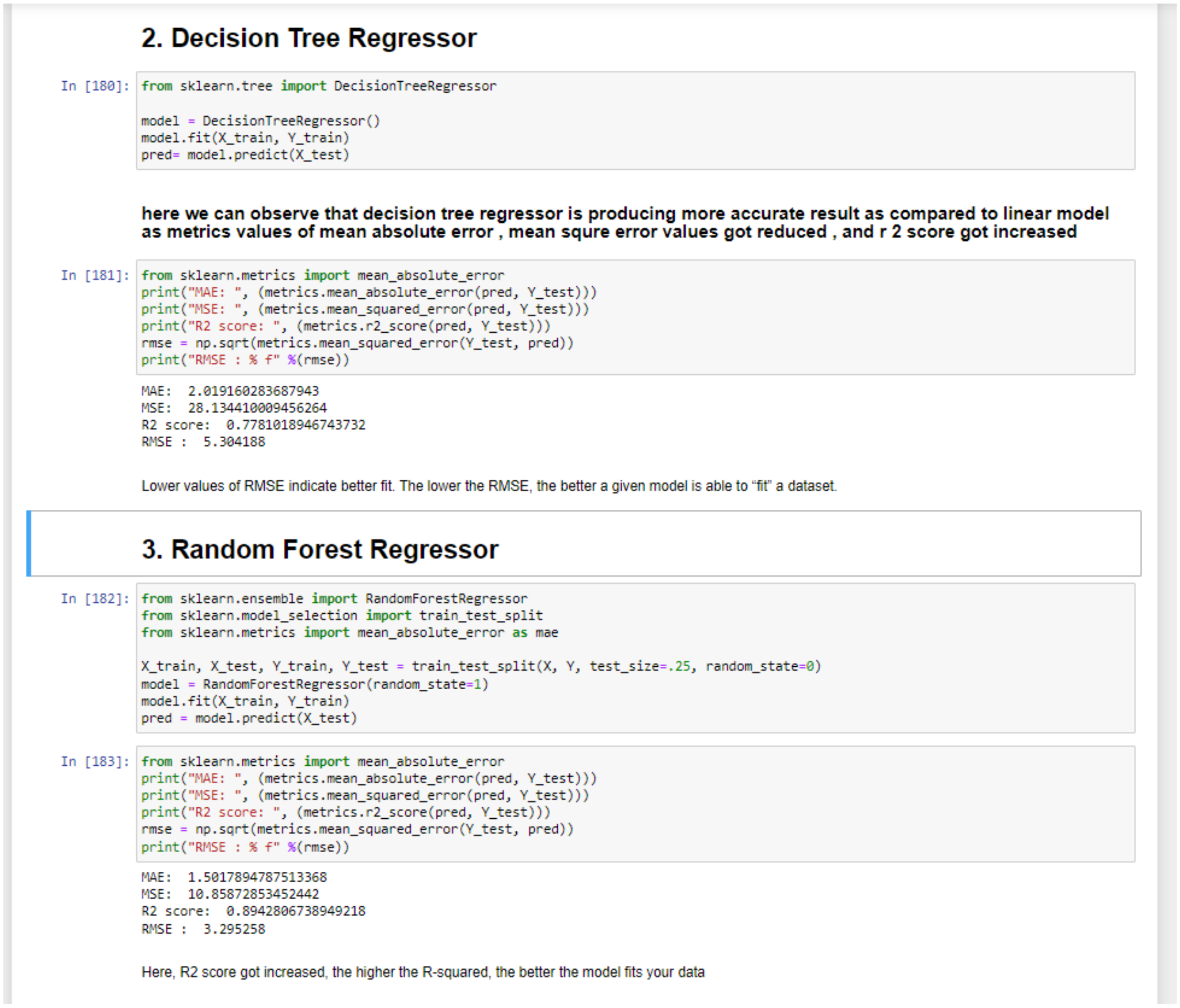
DECISION TREE REGRESSOR :-

Here we can see the model got fitted and we have metrices such as

5. MAE SCORE:- 2.01
6. MSE SCORE:- 28.13

7. R2 SCORE :- 0.77
8. RMSE:- 5.30

Here also we can infer that the built model has produced the above metrics and this score such as MAE, MSE, and RMSE is reducing ie on the other hand r2 score should be increasing let’s try some other model to reduce thse metrices and increase the r2 score .



2. Decision Tree Regressor

In [180]:
```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()
model.fit(X_train, Y_train)
pred= model.predict(X_test)
```

here we can observe that decision tree regressor is producing more accurate result as compared to linear model as metrics values of mean absolute error , mean squre error values got reduced , and r 2 score got increased

In [181]:
```
from sklearn.metrics import mean_absolute_error
print("MAE: ", (metrics.mean_absolute_error(pred, Y_test)))
print("MSE: ", (metrics.mean_squared_error(pred, Y_test)))
print("R2 score: ", (metrics.r2_score(pred, Y_test)))
rmse = np.sqrt(metrics.mean_squared_error(Y_test, pred))
print("RMSE : % f" %(rmse))
```

```
MAE:  2.019160283687943
MSE:  28.134410009456264
R2 score:  0.7781018946743732
RMSE :  5.304188
```

Lower values of RMSE indicate better fit. The lower the RMSE, the better a given model is able to “fit” a dataset.

3. Random Forest Regressor

In [182]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error as mae

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.25, random_state=0)
model = RandomForestRegressor(random_state=1)
model.fit(X_train, Y_train)
pred = model.predict(X_test)
```

In [183]:
```
from sklearn.metrics import mean_absolute_error
print("MAE: ", (metrics.mean_absolute_error(pred, Y_test)))
print("MSE: ", (metrics.mean_squared_error(pred, Y_test)))
print("R2 score: ", (metrics.r2_score(pred, Y_test)))
rmse = np.sqrt(metrics.mean_squared_error(Y_test, pred))
print("RMSE : % f" %(rmse))
```

```
MAE:  1.5017894787513368
MSE:  10.85872853452442
R2 score:  0.8942806738949218
RMSE :  3.295258
```

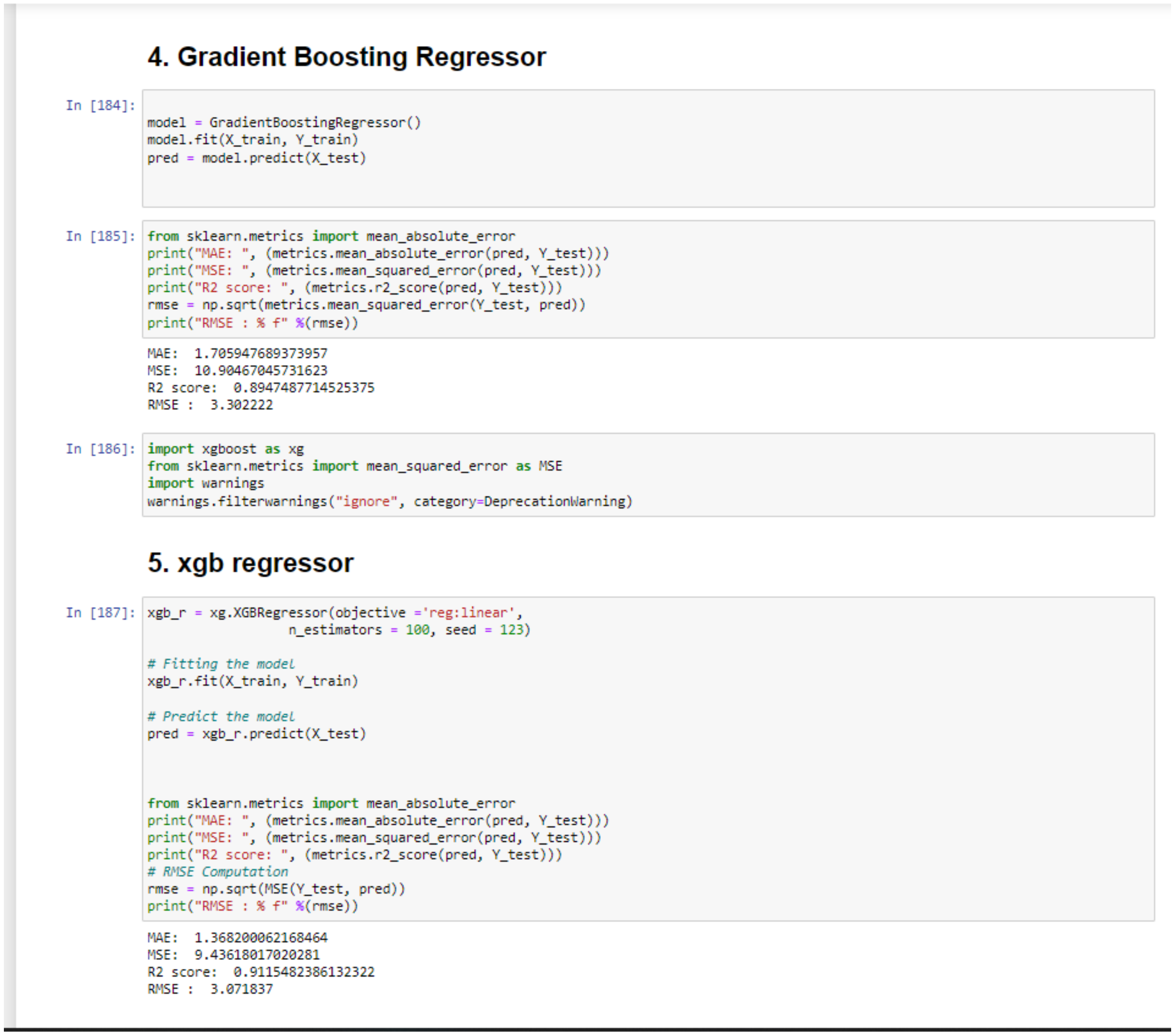
Here, R2 score got increased, the higher the R-squared, the better the model fits your data

RANDOM FOREST REGRESSOR:-

Here we can see the model got fitted and we have metrices such as

9. MAE SCORE:- 1.50
10. MSE SCORE:- 10.85
11. R2 SCORE:- 0.89
12. RMSE:- 3.29

Here also we can infer that the built model has produced the above metrics and this score such as MAE, MSE, and RMSE is reducing ie on the other hand r2 score should be increasing let’s try some other model to reduce thse metrices and increase the r2 score .



4. Gradient Boosting Regressor

In [184]:
```
model = GradientBoostingRegressor()
model.fit(X_train, Y_train)
pred = model.predict(X_test)
```

In [185]:
```
from sklearn.metrics import mean_absolute_error
print("MAE: ", (metrics.mean_absolute_error(pred, Y_test)))
print("MSE: ", (metrics.mean_squared_error(pred, Y_test)))
print("R2 score: ", (metrics.r2_score(pred, Y_test)))
rmse = np.sqrt(metrics.mean_squared_error(Y_test, pred))
print("RMSE : % f" %(rmse))
```

```
MAE:  1.705947689373957
MSE:  10.90467045731623
R2 score:  0.8947487714525375
RMSE :  3.302222
```

In [186]:
```
import xgboost as xg
from sklearn.metrics import mean_squared_error as MSE
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

5. xgb regressor

In [187]:
```
xgb_r = xg.XGBRegressor(objective ='reg:linear',
                  n_estimators = 100, seed = 123)

# Fitting the model
xgb_r.fit(X_train, Y_train)

# Predict the model
pred = xgb_r.predict(X_test)


from sklearn.metrics import mean_absolute_error
print("MAE: ", (metrics.mean_absolute_error(pred, Y_test)))
print("MSE: ", (metrics.mean_squared_error(pred, Y_test)))
print("R2 score: ", (metrics.r2_score(pred, Y_test)))
# RMSE Computation
rmse = np.sqrt(MSE(Y_test, pred))
print("RMSE : % f" %(rmse))
```

```
MAE:  1.368200062168464
MSE:  9.43618017020281
R2 score:  0.9115482386132322
RMSE :  3.071837
```

GRADIENT BOOSTING REGRESSOR:-

Here we can see the model got fitted and we have metrices such as

13.MAE SCORE:- 1.50
14.MSE SCORE:- 10.90
15.R2 SCORE:- 0.84
16.RMSE:- 3.30

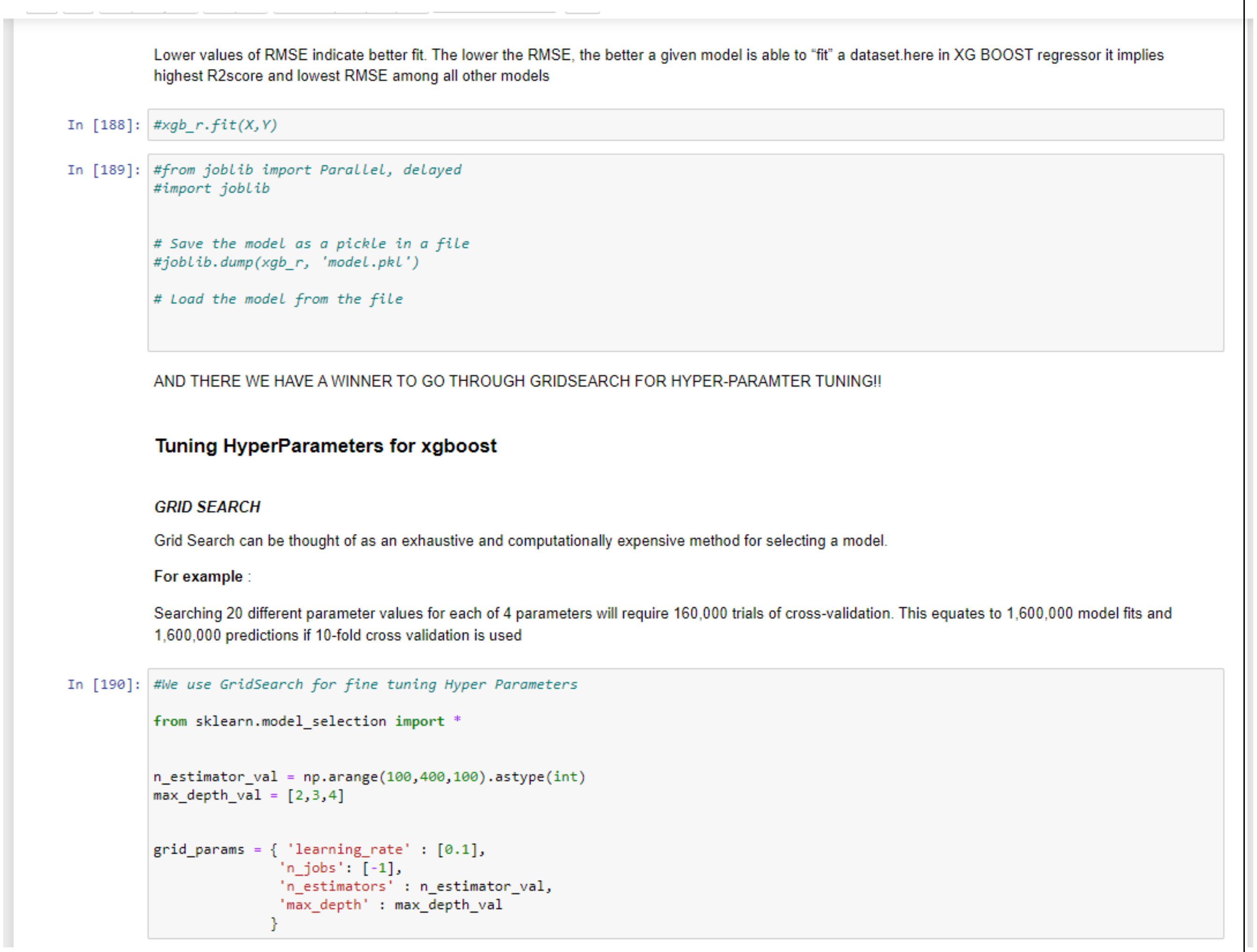
Here also we can infer that the built model has produced the above metrics and this score such as MAE, MSE, and RMSE is reducing ie on the other hand r2 score should be increasing let’s try the model to reduce the metrics and increase the r2 score

XG BOOST REGRESSOR:-

Here we can see the model got fitted and we have metrics such as

17.MAE SCORE:- 1.36
18.MSE SCORE:- 9.43
19.R2 SCORE:- 0.91
20.RMSE:- 3.29

Here also we can infer that the built model has produced the above metrics and these scores such as MAE, MSE, and RMSE reached to the minimum value as compared to ie on the other hand r2 score is also high as compared to all let’s try some other model to reduce thse metrices and increase the r2 score



Lower values of RMSE indicate better fit. The lower the RMSE, the better a given model is able to "fit" a dataset.here in XG BOOST regressor it implies highest R2score and lowest RMSE among all other models

```
In [188]: #xgb_r.fit(X,Y)
```

```
In [189]: #from joblib import Parallel, delayed
          #import joblib


          # Save the model as a pickle in a file
          #joblib.dump(xgb_r, 'model.pkl')

          # Load the model from the file
```

AND THERE WE HAVE A WINNER TO GO THROUGH GRIDSEARCH FOR HYPER-PARAMTER TUNING!!

Tuning HyperParameters for xgboost

GRID SEARCH

Grid Search can be thought of as an exhaustive and computationally expensive method for selecting a model.

For example :

Searching 20 different parameter values for each of 4 parameters will require 160,000 trials of cross-validation. This equates to 1,600,000 model fits and 1,600,000 predictions if 10-fold cross validation is used

```
In [190]: #We use GridSearch for fine tuning Hyper Parameters

          from sklearn.model_selection import *


          n_estimator_val = np.arange(100,400,100).astype(int)
          max_depth_val = [2,3,4]


          grid_params = { 'learning_rate' : [0.1],
                          'n_jobs': [-1],
                          'n_estimators' : n_estimator_val,
                          'max_depth' : max_depth_val
                        }
```

- Here wehave done hyper parameter tuning using grid search and reandom random method
- We have found the beast parameter such as learning rate =0.1, max depth =4,no. of estimator =300,no_jobs=1.
- And same parameter we have observed from random search algortihm which is mentioned below

To Display the Best Score

```
In [192]: gs_results.best_score_
```

```
Out[192]: 0.9071962570393127
```

To Display the Best Estimator

```
In [193]: gs_results.best_estimator_
```

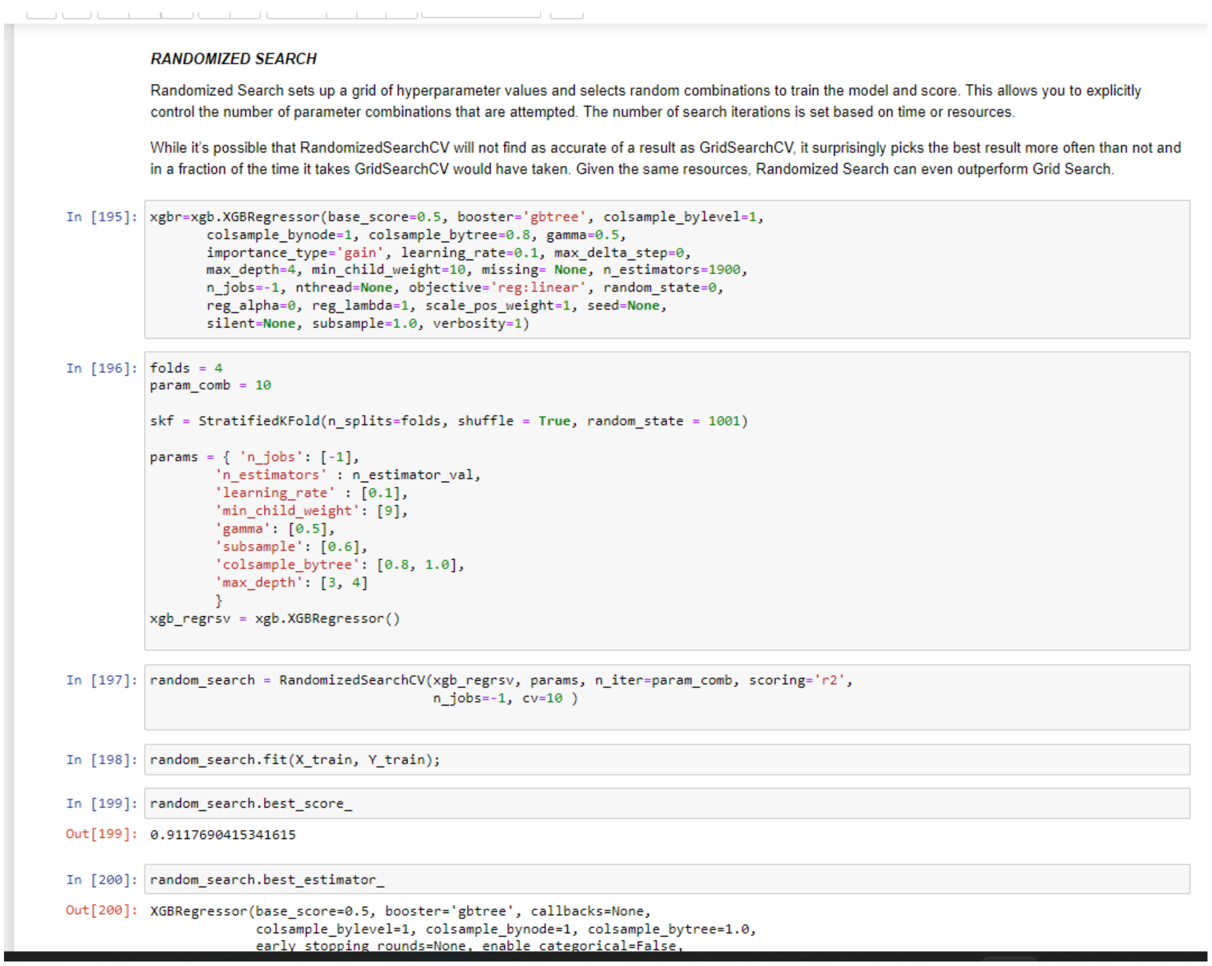
```
Out[193]: XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
                       colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                       early_stopping_rounds=None, enable_categorical=False,
                       eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
                       importance_type=None, interaction_constraints='',
                       learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
                       max_delta_step=0, max_depth=4, max_leaves=0, min_child_weight=1,
                       missing=nan, monotone_constraints='()', n_estimators=300,
                       n_jobs=-1, num_parallel_tree=1, predictor='auto', random_state=0,
                       reg_alpha=0, reg_lambda=1, ...)
```

To Display the Best Parameters

```
In [194]: gs_results.best_params_
```

```
Out[194]: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 300, 'n_jobs': -1}
```

We could settle here but let's try randomized search cross validation.


RANDOMIZED SEARCH

Randomized Search sets up a grid of hyperparameter values and selects random combinations to train the model and score. This allows you to explicitly control the number of parameter combinations that are attempted. The number of search iterations is set based on time or resources.

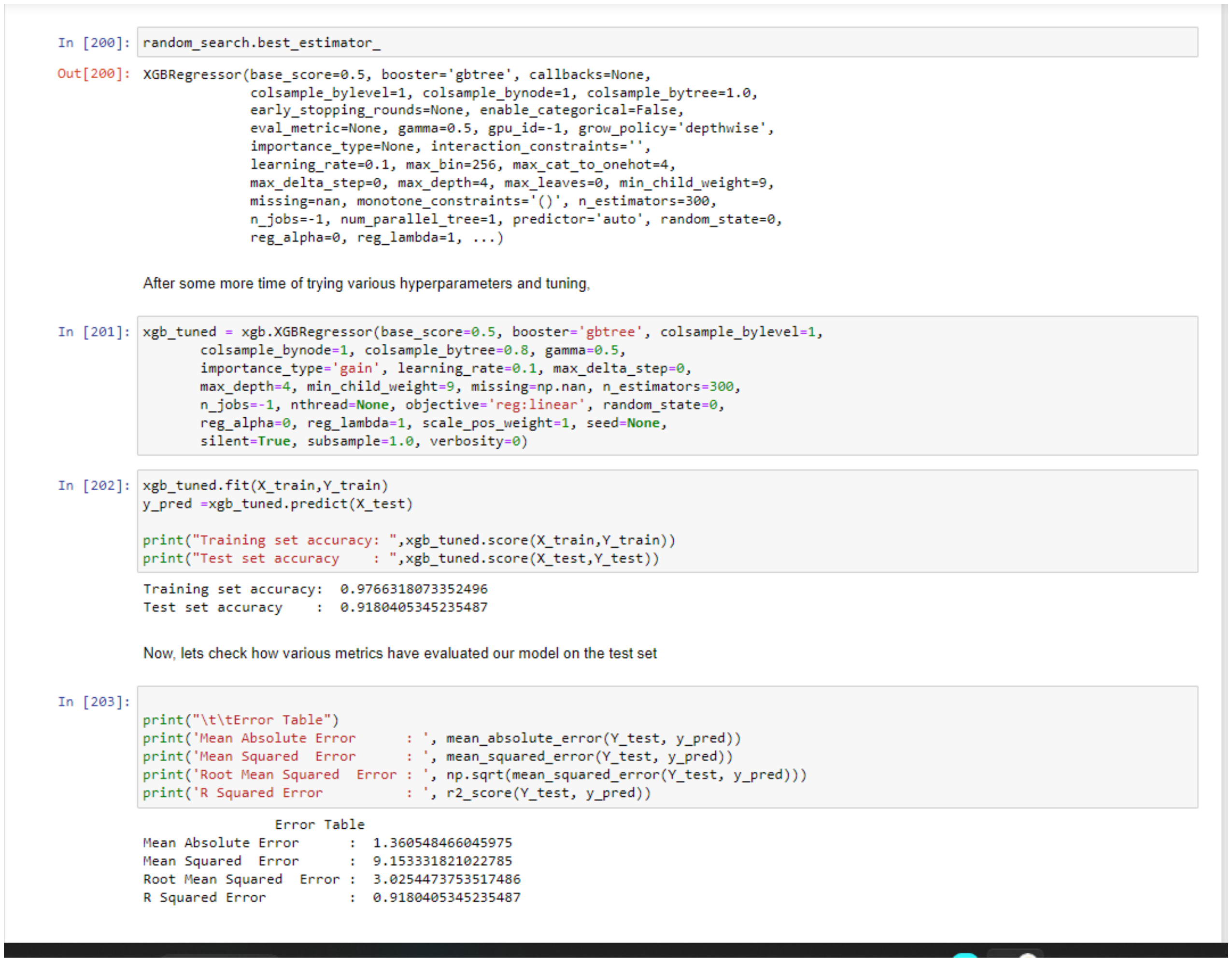
While it's possible that RandomizedSearchCV will not find as accurate of a result as GridSearchCV, it surprisingly picks the best result more often than not and in a fraction of the time it takes GridSearchCV would have taken. Given the same resources, Randomized Search can even outperform Grid Search.

```
In [195]: xgbr=xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                 colsample_bynode=1, colsample_bytree=0.8, gamma=0.5,
                 importance_type='gain', learning_rate=0.1, max_delta_step=0,
                 max_depth=4, min_child_weight=10, missing= None, n_estimators=1900,
                 n_jobs=-1, nthread=None, objective='reg:linear', random_state=0,
                 reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                 silent=None, subsample=1.0, verbosity=1)

In [196]: folds = 4
          param_comb = 10

          skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

          params = { 'n_jobs': [-1],
                  'n_estimators' : n_estimator_val,
                  'learning_rate' : [0.1],
                  'min_child_weight': [9],
                  'gamma': [0.5],
                  'subsample': [0.6],
                  'colsample_bytree': [0.8, 1.0],
                  'max_depth': [3, 4]
                  }
          xgb_regrsv = xgb.XGBRegressor()

In [197]: random_search = RandomizedSearchCV(xgb_regrsv, params, n_iter=param_comb, scoring='r2',
                                             n_jobs=-1, cv=10 )

In [198]: random_search.fit(X_train, Y_train);

In [199]: random_search.best_score_

Out[199]: 0.9117690415341615

In [200]: random_search.best_estimator_

Out[200]: XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
                       colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1.0,
                       early_stopping_rounds=None, enable_categorical=False,
```

- After finding the best parameter we tuned it according to that parameter to get the best accuracy
- And we have also drawn the matrices and accuracy on the training set and testing set of the data
  1. Training set score =97%
  2. Testing set score=91%

- Metrics
  3. MAE SCORE:- 1.36
  4. MSE SCORE:- 9.15
  5. R2 SCORE:- 0.91
  6. RMSE:- 3.02

These is the minimum metrics that we have drawn among all model and with the xg boost also , before tuning the r2 score was less from 0.91 and paralley other score like MAE,MSE,RMSE we also little high



```
In [200]: random_search.best_estimator_

Out[200]: XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
                       colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1.0,
                       early_stopping_rounds=None, enable_categorical=False,
                       eval_metric=None, gamma=0.5, gpu_id=-1, grow_policy='depthwise',
                       importance_type=None, interaction_constraints='',
                       learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
                       max_delta_step=0, max_depth=4, max_leaves=0, min_child_weight=9,
                       missing=nan, monotone_constraints='()', n_estimators=300,
                       n_jobs=-1, num_parallel_tree=1, predictor='auto', random_state=0,
                       reg_alpha=0, reg_lambda=1, ...)
```

After some more time of trying various hyperparameters and tuning,

```
In [201]: xgb_tuned = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                 colsample_bynode=1, colsample_bytree=0.8, gamma=0.5,
                 importance_type='gain', learning_rate=0.1, max_delta_step=0,
                 max_depth=4, min_child_weight=9, missing=np.nan, n_estimators=300,
                 n_jobs=-1, nthread=None, objective='reg:linear', random_state=0,
                 reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                 silent=True, subsample=1.0, verbosity=0)
```

```
In [202]: xgb_tuned.fit(X_train,Y_train)
          y_pred =xgb_tuned.predict(X_test)

          print("Training set accuracy: ",xgb_tuned.score(X_train,Y_train))
          print("Test set accuracy    : ",xgb_tuned.score(X_test,Y_test))

          Training set accuracy:  0.9766318073352496
          Test set accuracy    :  0.9180405345235487
```

Now, lets check how various metrics have evaluated our model on the test set

```
In [203]:
          print("\t\tError Table")
          print('Mean Absolute Error      : ', mean_absolute_error(Y_test, y_pred))
          print('Mean Squared  Error      : ', mean_squared_error(Y_test, y_pred))
          print('Root Mean Squared  Error : ', np.sqrt(mean_squared_error(Y_test, y_pred)))
          print('R Squared Error          : ', r2_score(Y_test, y_pred))

                          Error Table
          Mean Absolute Error      :  1.360548466045975
          Mean Squared  Error      :  9.153331821022785
          Root Mean Squared  Error :  3.0254473753517486
          R Squared Error          :  0.9180405345235487
```

- CROSS-VALIDATION:-

  After doing Hyper paramet tuning we have applied the cross validation on our and got the accuracy scrore of 84.26%

cross validation strategy

Cross Validation is used to assess the predictive performance of the models and and to judge how they perform outside the sample to a new data set also known as test data The motivation to use cross validation techniques is that when we fit a model, we are fitting it to a training dataset. Without cross validation we only have information on how does our model perform to our in-sample data. Ideally we would like to see how does the model perform when we have a new data in terms of accuracy of its predictions. In science, theories are judged by its predictive performance.

We use the cross_val_score function of Sklearn. However this function has not a shuffle attribut, we add then one line of code, in order to shuffle the dataset prior to cross-validation

In [208]:
```
#Validation function
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(X_train.values)
    rmse= np.sqrt(-cross_val_score(model, X_train.values, y_train.values, scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

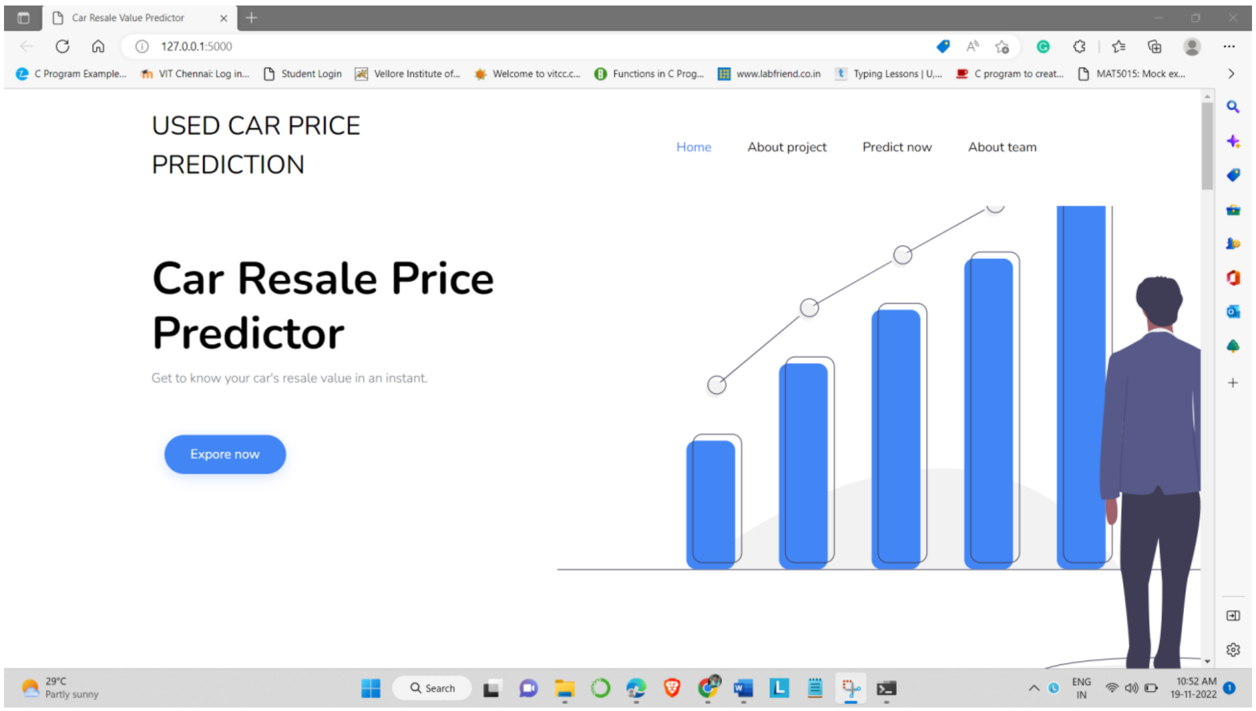
In [209]:
```
from sklearn.model_selection import KFold, cross_val_score, train_test_split
random_model = RandomForestRegressor(n_estimators=300, max_depth=4)
#Fit
random_model.fit(X_train, Y_train)
xgb_accuracy = round(random_model.score(X_train,Y_train)*100,2)
print(round(xgb_accuracy,2),'%')
```

```
84.26 %
```

CONCLUSION

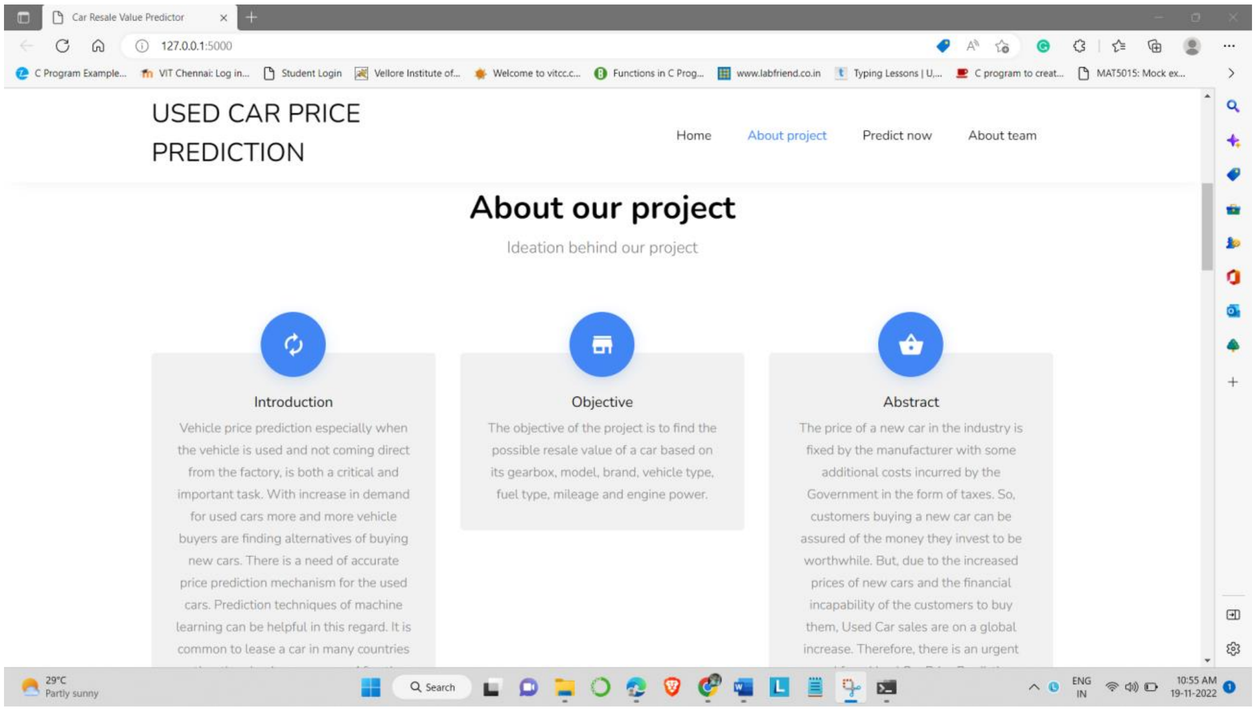
This model is ready to be deployed in a pipeline.
We have accomplished the task of building a good model for our used cars' prediction purposes.

- **Our Website :-**



Car Resale Price Predictor

Get to know your car's resale value in an instant.



USED CAR PRICE PREDICTION

Home About project Predict now About team

About our project

Ideation behind our project

Introduction

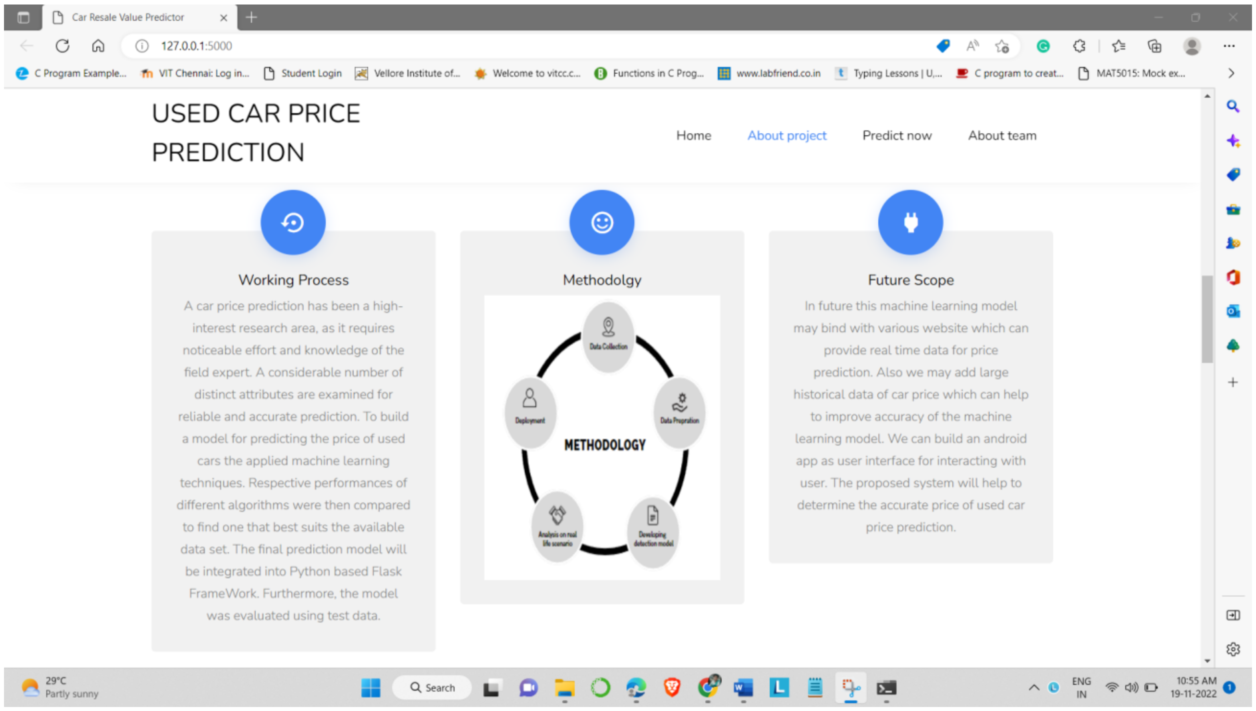
Vehicle price prediction especially when the vehicle is used and not coming direct from the factory, is both a critical and important task. With increase in demand for used cars more and more vehicle buyers are finding alternatives of buying new cars. There is a need of accurate price prediction mechanism for the used cars. Prediction techniques of machine learning can be helpful in this regard. It is common to lease a car in many countries

Objective

The objective of the project is to find the possible resale value of a car based on its gearbox, model, brand, vehicle type, fuel type, mileage and engine power.

Abstract

The price of a new car in the industry is fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So, customers buying a new car can be assured of the money they invest to be worthwhile. But, due to the increased prices of new cars and the financial incapability of the customers to buy them, Used Car sales are on a global increase. Therefore, there is an urgent



USED CAR PRICE PREDICTION

Home About project Predict now About team

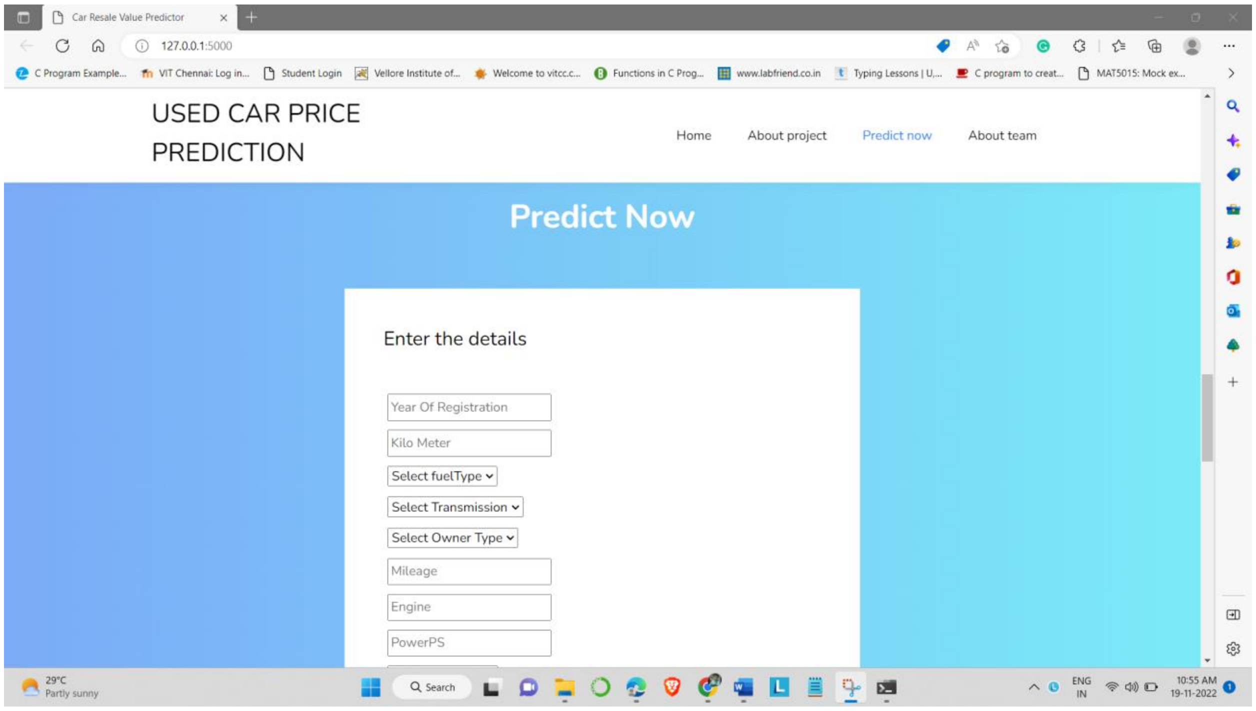
Working Process

A car price prediction has been a high-interest research area, as it requires noticeable effort and knowledge of the field expert. A considerable number of distinct attributes are examined for reliable and accurate prediction. To build a model for predicting the price of used cars the applied machine learning techniques. Respective performances of different algorithms were then compared to find one that best suits the available data set. The final prediction model will be integrated into Python based Flask FrameWork. Furthermore, the model was evaluated using test data.

Methodolgy

Future Scope

In future this machine learning model may bind with various website which can provide real time data for price prediction. Also we may add large historical data of car price which can help to improve accuracy of the machine learning model. We can build an android app as user interface for interacting with user. The proposed system will help to determine the accurate price of used car price prediction.


USED CAR PRICE PREDICTION

Home About project Predict now About team

Predict Now

Enter the details

Year Of Registration

Kilo Meter

Select fuelType

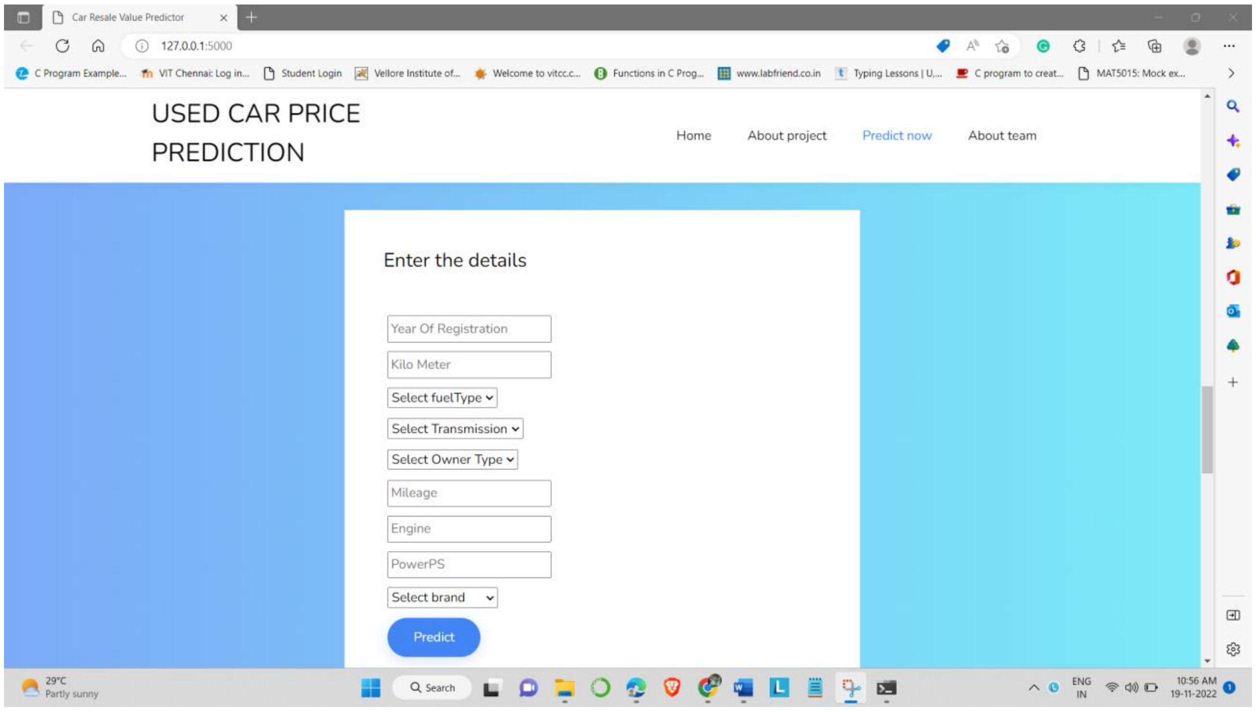
Select Transmission

Select Owner Type

Mileage

Engine

PowerPS


USED CAR PRICE PREDICTION

Home About project Predict now About team

Enter the details

Year Of Registration

Kilo Meter

Select fuelType

Select Transmission

Select Owner Type

Mileage

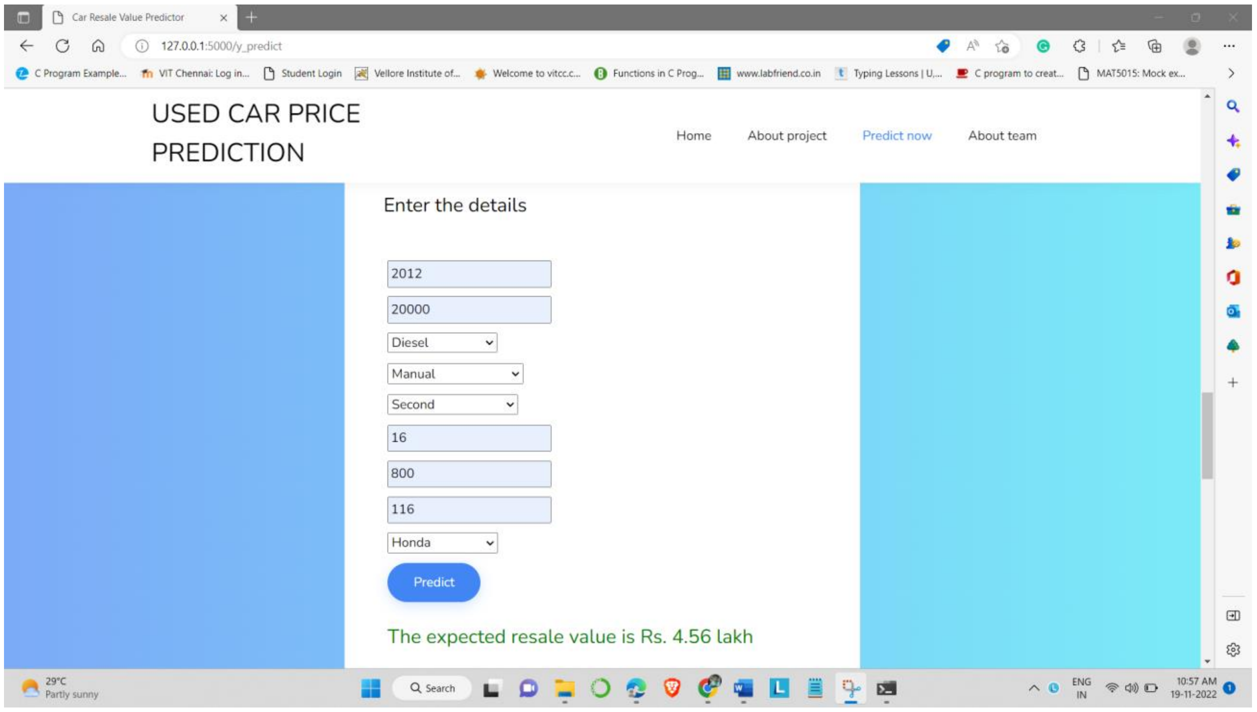
Engine

PowerPS

Select brand

Predict

PREDICTION:-



Car Resale Value Predictor

127.0.0.1:5000/y_predict

USED CAR PRICE PREDICTION

Home | About project | Predict now | About team

Enter the details

2012

20000

Diesel

Manual

Second

16

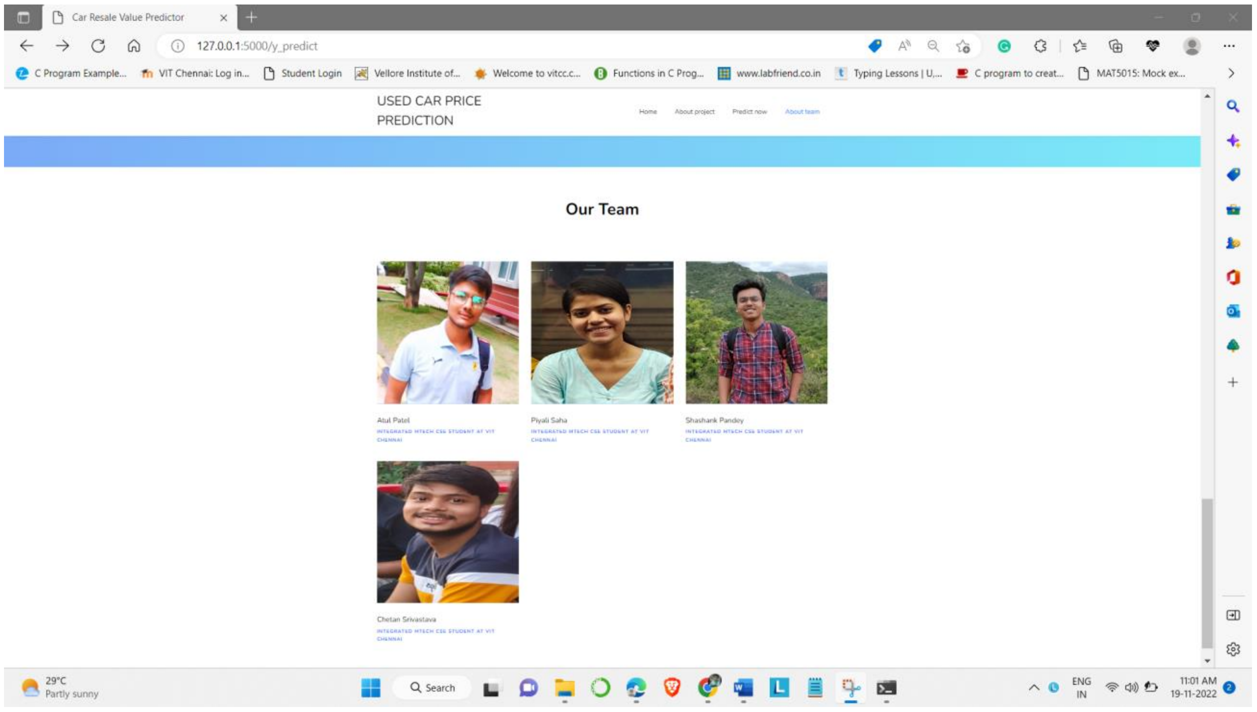
800

116

Honda

Predict

The expected resale value is Rs. 4.56 lakh



Car Resale Value Predictor

127.0.0.1:5000/y_predict

USED CAR PRICE PREDICTION

Home | About project | Predict now | About team

Our Team

Atul Patel
INTEGRATED MTECH CSE STUDENT AT VIT CHENNAI

Piyali Saha
INTEGRATED MTECH CSE STUDENT AT VIT CHENNAI

Shashank Pandey
INTEGRATED MTECH CSE STUDENT AT VIT CHENNAI

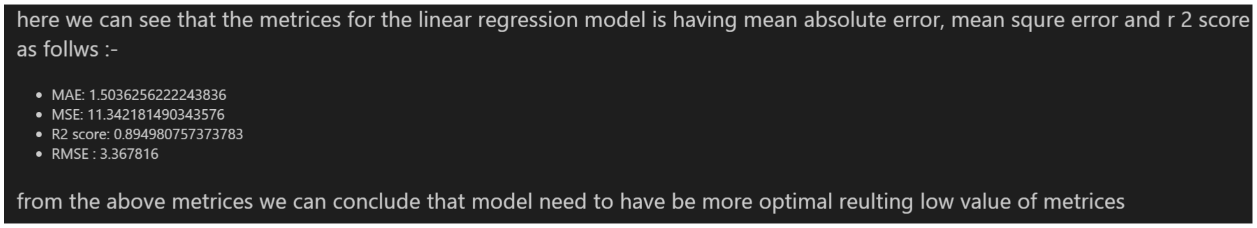
Chetan Srivastava
INTEGRATED MTECH CSE STUDENT AT VIT CHENNAI

Results and Discussion

Results showing testing, metrics taken for checking and a complete analysis on the results obtained.

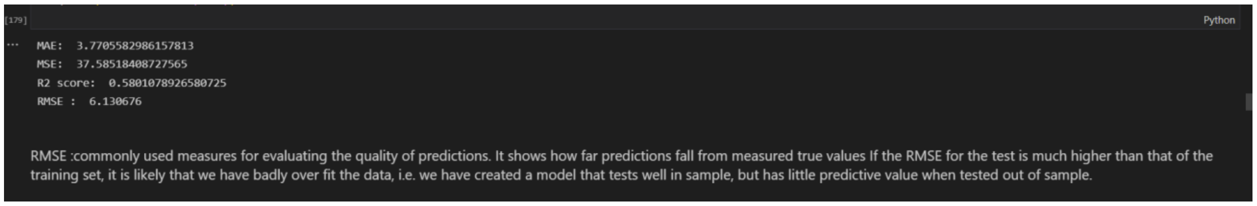
From the above

- **Linear Regression Model: -**


here we can see that the metrices for the linear regression model is having mean absolute error, mean squre error and r 2 score as follws :-

- MAE: 1.5036256222243836
- MSE: 11.342181490343576
- R2 score: 0.894980757373783
- RMSE : 3.367816

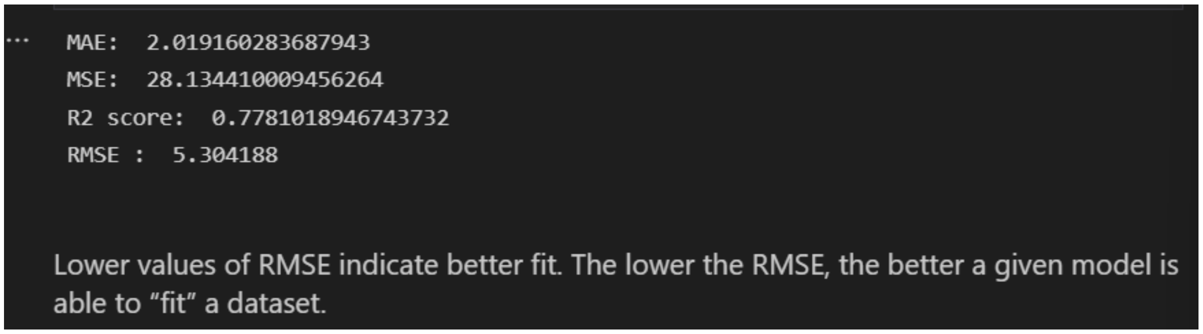
from the above metrices we can conclude that model need to have be more optimal reulting low value of metrices


[179] Python

```
···  MAE:  3.7705582986157813
     MSE:  37.58518408727565
     R2 score:  0.5801078926580725
     RMSE :  6.130676
```

RMSE :commonly used measures for evaluating the quality of predictions. It shows how far predictions fall from measured true values If the RMSE for the test is much higher than that of the training set, it is likely that we have badly over fit the data, i.e. we have created a model that tests well in sample, but has little predictive value when tested out of sample.

- **Decision Tree Regressor: -**

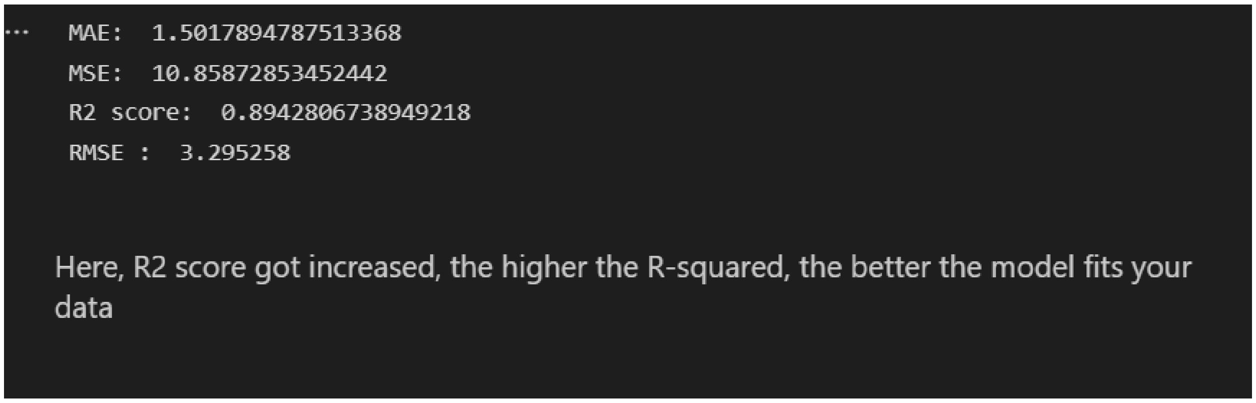


```
···  MAE:  2.019160283687943
     MSE:  28.134410009456264
     R2 score:  0.7781018946743732
     RMSE :  5.304188
```

Lower values of RMSE indicate better fit. The lower the RMSE, the better a given model is able to “fit” a dataset.

here we can observe that decision tree regressor is producing more accurate result as compared to linear model as metrics values of mean absolute error, mean square error values got reduced , and r 2 score got increased

- **Radom Tree Regressor: -**

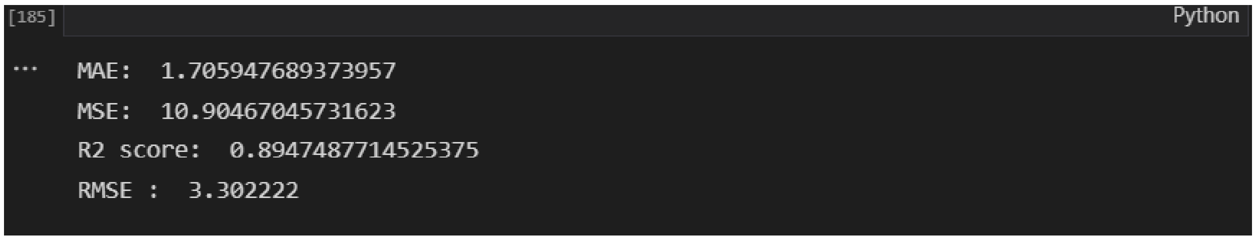


```
...   MAE:  1.5017894787513368
      MSE:  10.85872853452442
      R2 score:  0.8942806738949218
      RMSE :  3.295258
```

Here, R2 score got increased, the higher the R-squared, the better the model fits your data

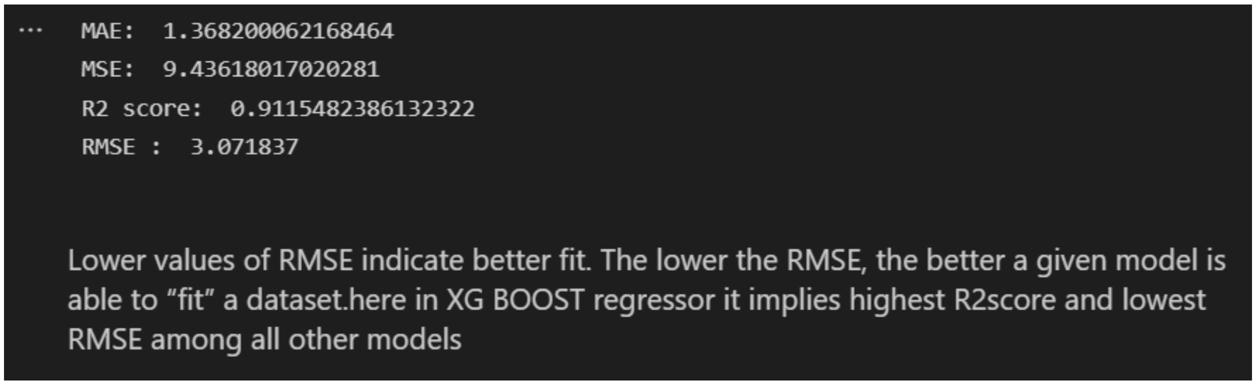
- **Gradient Boosting Regressor: -**

=



```
[185]                                                    Python
...   MAE:  1.705947689373957
      MSE:  10.90467045731623
      R2 score:  0.8947487714525375
      RMSE :  3.302222
```

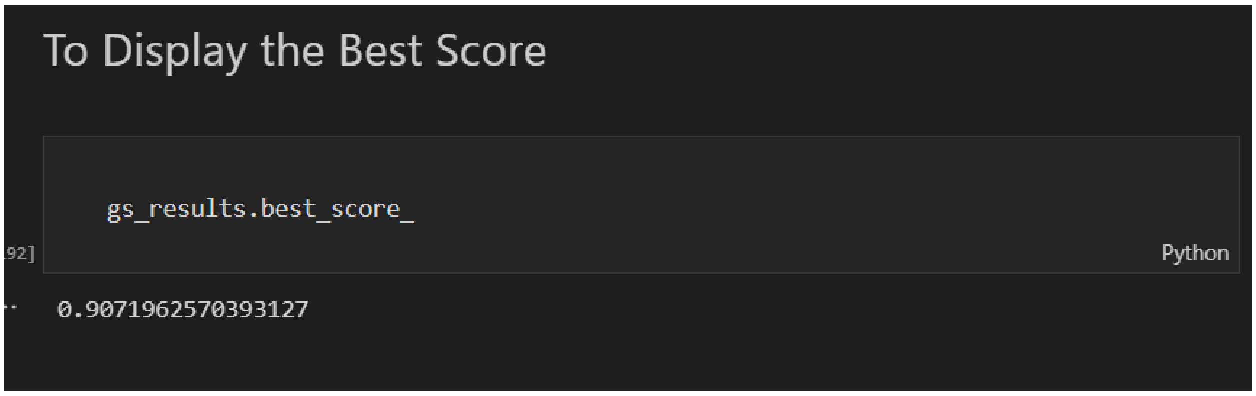
- **XGB Regressor: -**



```
···  MAE:  1.368200062168464
     MSE:  9.43618017020281
     R2 score:  0.9115482386132322
     RMSE :  3.071837
```

Lower values of RMSE indicate better fit. The lower the RMSE, the better a given model is able to “fit” a dataset.here in XG BOOST regressor it implies highest R2score and lowest RMSE among all other models

- **Hyper Parameter Tuning the Best Score: -**

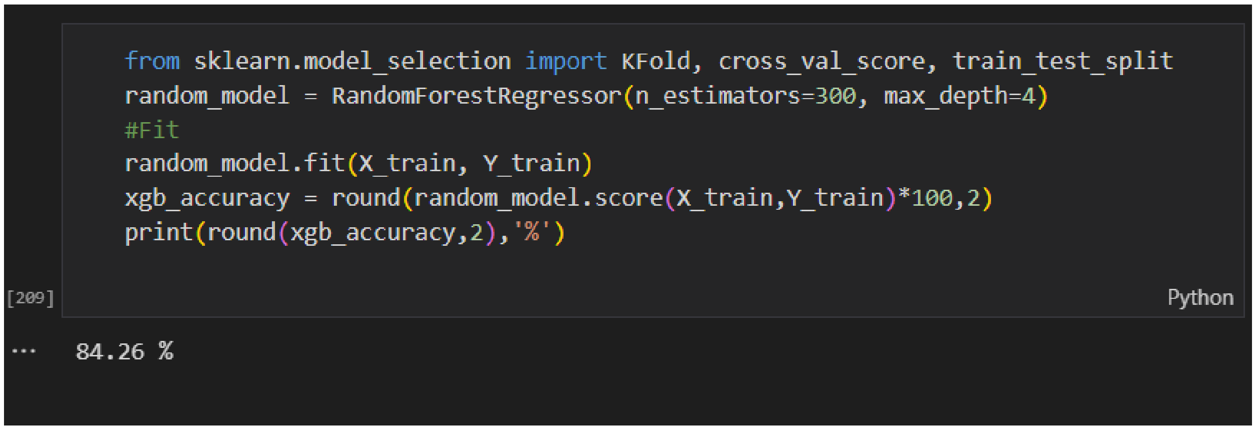


To Display the Best Score

```
gs_results.best_score_
```

```
0.9071962570393127
```

- **Cross Validation: -**



```
from sklearn.model_selection import KFold, cross_val_score, train_test_split
random_model = RandomForestRegressor(n_estimators=300, max_depth=4)
#Fit
random_model.fit(X_train, Y_train)
xgb_accuracy = round(random_model.score(X_train,Y_train)*100,2)
print(round(xgb_accuracy,2),'%')
```

```
···  84.26 %
```

- Here we have concluded that after analyzing the parameter and the matrices of all the models we have used that XG Boost Regressor is the best Model among all.

- As we have applied the hyper parameter tunning and derived the best parameter with the best score of 90%

- Also, we have done the Cross Validation Of the model to Cross verify the Models and we got XG Boost regressor as the highest scoring model when compared with other

Conclusion and Future works

The increased prices of new cars and the financial incapability of the customers to buy them, Used Car sales are on a global increase. Therefore, there is an urgent need for a Used Car Price Prediction system which effectively determines the worthiness of the car using a variety of features. The proposed system will help to determine the accurate price of used car price prediction. This paper compares 3 different algorithms for machine learning: Linear Regression, random forest regression and XG Boost Regression.

In future this machine learning model may bind with various website which can provide real time data for price prediction. Also, we may add large historical data of car price which can help to improve accuracy of the machine learning model. We can build an android app as user interface for interacting with user. For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset.

References

- https://www.ijraset.com/research-paper/price-prediction-of-used-cars-using-machine-learning
- https://www.researchgate.net/publication/343878698_Used_Cars_Price_Prediction_using_Supervised_Learning_Techniques
- https://www.irjet.net/archives/V8/i4/IRJET-V8I4278.pdf
- https://towardsdatascience.com/used-car-price-prediction-using-machine-learning-e3be02d977b2