MAJOR PROJECT REPORT

TOPIC :

CAR PRICE PREDICTION MODEL

SUBMITTED BY :

Palash Mishra

Jaypee University of Engineering and Technology, Guna

OVERVIEW:

CAR SELLING AND BUYING PROBLEM -

To Predict the Sell Price for the car. This problem is a regression problem.

Predict the best equation for Sell Price for the car using the available dataset "CAR DATA" from Kaggle, in which the car details have been given by "car Dekho".

Aim of the project:

1. Download data 'car data.csv'.
2. Split the data to train and to test method
3. Take the useful features to predict the equation
4. Train the regressor so that it can give the equation to find sell price of car
5. Test the model by the data.
6. To find lose functions.

Data exploration and details:

This dataset contains information about used cars.It has been taken from https://www.kaggle.com/nehalbirla/vehicle-dataset-from-cardekho

This data can be used for a lot of purposes such as price prediction to exemplify the use of linear regression in Machine Learning.

The columns in the given dataset are as follows:

name

year

selling_price

km_driven

fuel

seller_type

transmission

Owner

Details of features given :

1. Name : Name of the cars
2. Year : Year of the car when it was bought
3. Selling_Price : Price at which the car is being sold
4. Kms_driven : Number of Kilometres the car is driven
5. fuel : Fuel type of car (petrol / diesel / CNG / LPG / electric)
6. Seller_Type : Tells if a Seller is Individual or a Dealer
7. Transmission : Gear transmission of the car (Automatic/Manual)
8. Owner : Number of previous owners of the car.

In this dataset , we have added a few features and also deleted a few.

Added features :

1. current_year : we added the ongoing year

2. no_of_year : this calculates the how many year the car is old.

It is calculated using = current year - year

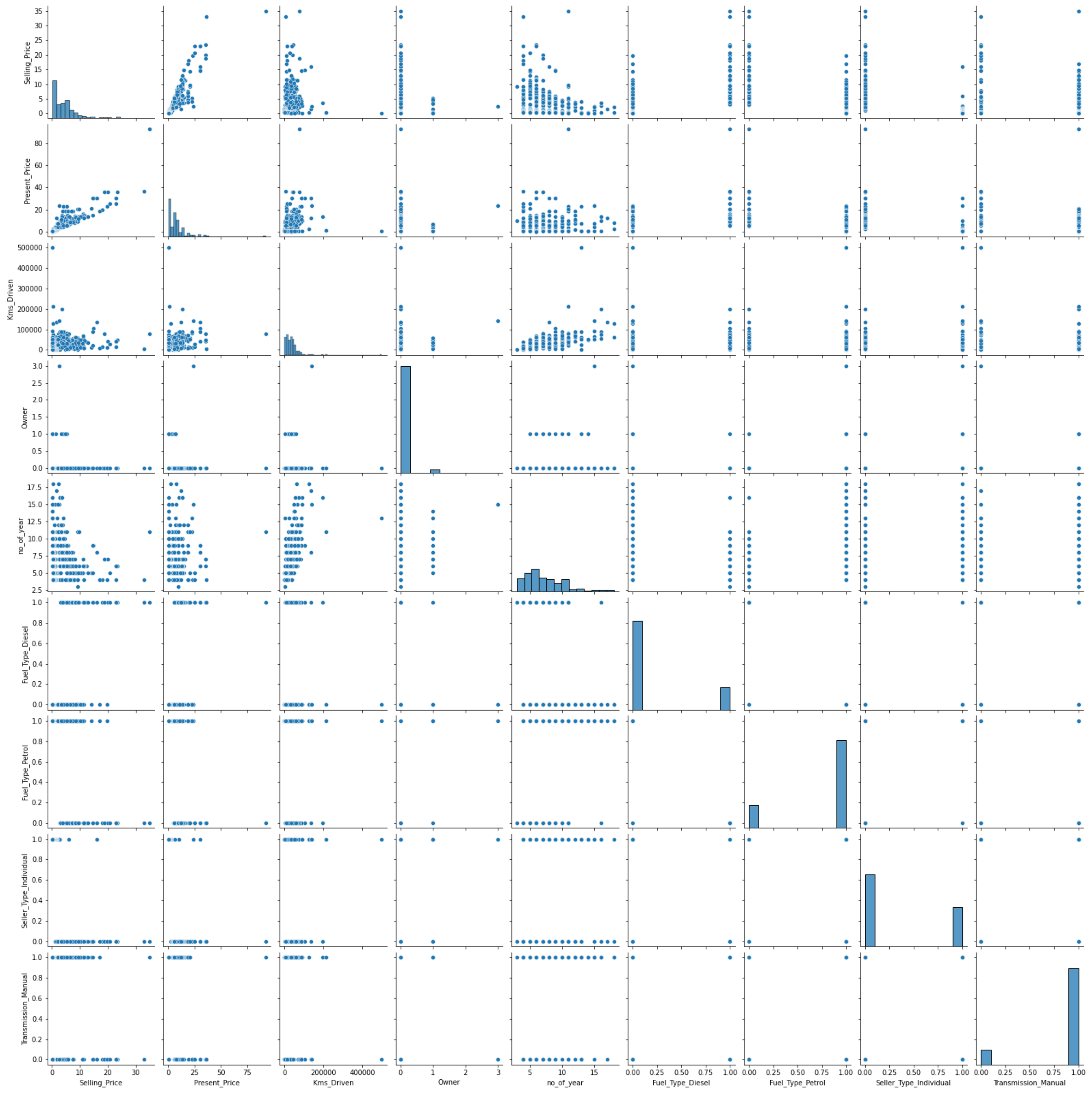
Columns deleted :

1. current_year: As it was taken to calculate no_of_year

2. Year : Taken in use to calculate no_of_year

3. name : It is not helpful in predicting the price

So, we select the final data sets and following graph can be used to see the final datasets.



(pairplot)

Algorithms and Methods used :

The regressor/model used is a Random Forest Regressor , A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Ensemble methods:

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

The sklearn.ensemble module includes two averaging algorithms based on randomized decision trees: the RandomForest algorithm and the Extra-

Trees method. Both algorithms are perturb-and-combine techniques [B1998] specifically designed for trees.

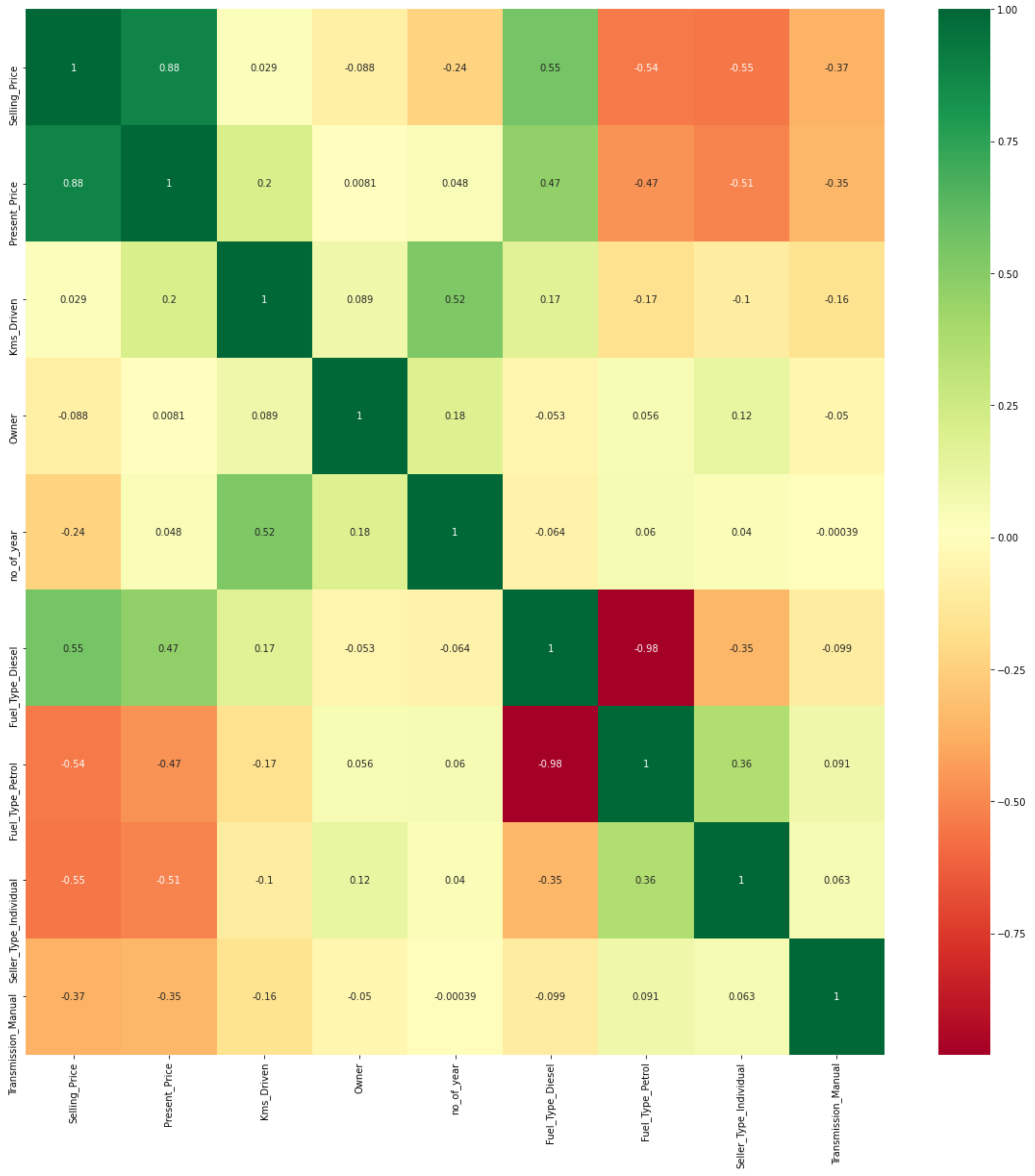
Source:https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees

Parameters and hyperparameters :

Correlated features will be given equal or similar importance, but overall reduced importance compared to the same tree built without correlated counterparts. Random Forests and decision trees, in general, give preference to features with high cardinality (Trees are biased to these type of variables).

Source : https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Showing Correlation between feautres using heatmap :



Feature importance evaluation :

The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable. Features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features. In scikit-learn, the fraction of samples a feature contributes to is combined with the decrease in impurity from splitting them to create a normalized estimate of the predictive power of that feature.

Feature selection:

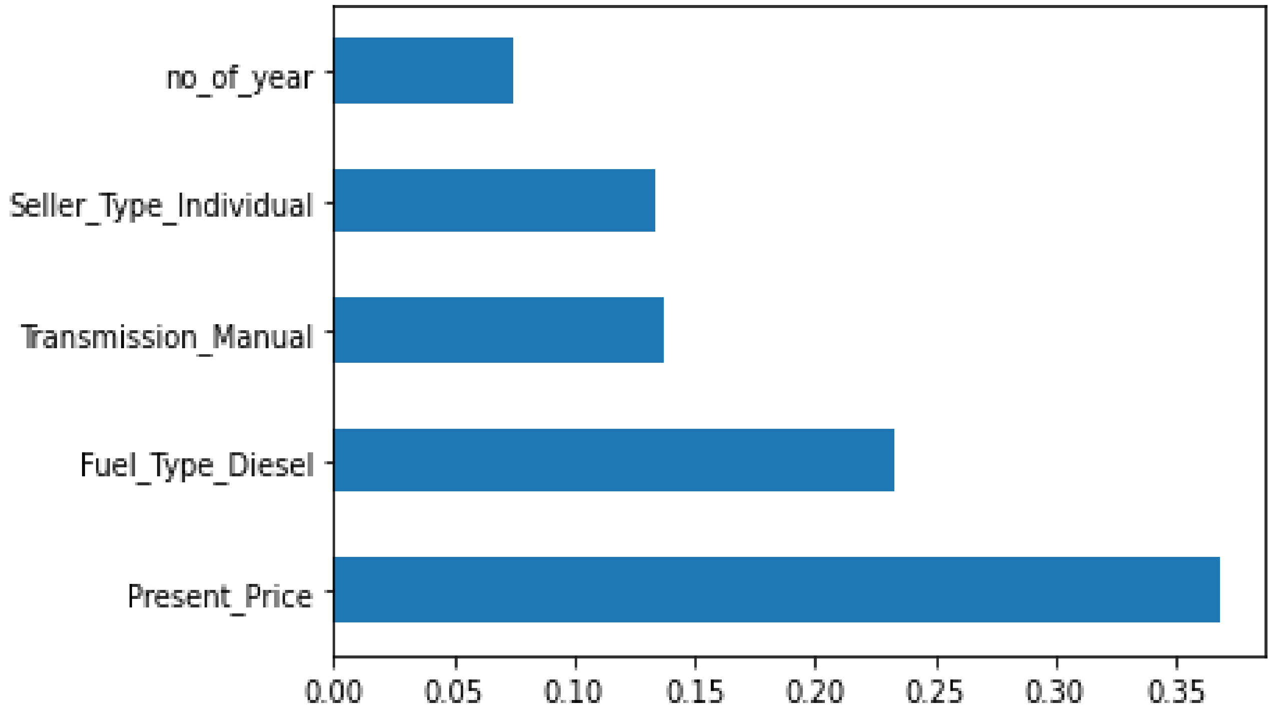
The classes in the sklearn.feature_selection module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

Tree-based feature selection:

Tree-based estimators (see the sklearn.tree module and forest of trees in the sklearn.ensemble module) can be used to compute impurity-based feature importances, which in turn can be used to discard irrelevant features (when coupled with the SelectFromModel meta-transformer).

Source:https://scikit-learn.org/stable/modules/feature_selection.html

https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e



(feature importance graph)

Hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance

While model parameters are learned during training — such as the slope and intercept in a linear regression — hyperparameters must be set by the data scientist before training. In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. (The parameters of a random forest are the variables and thresholds used to split each node learned during training). Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error based engineering.

Cross Validation:

The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data).

The main parameters to adjust when using these methods is n_estimators and max_features. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are max_features=None (always considering all features instead of a random subset) for regression problems, and max_features="sqrt" (using a random subset of size sqrt(n_features)) for classification tasks (where n_features is the number of features in the data). Good results are often achieved when setting max_depth=None in combination with min_samples_split=2 (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated.

We tried adjusting the following set of hyperparameters:

n_estimators = number of trees in the foreset

max_features = max number of features considered for splitting a node

max_depth = max number of levels in each decision tree min_samples_split = min number of data points placed in a node before the node is split

min_samples_leaf = min number of data points allowed in a leaf node

Using Scikit-Learn’s RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold CV with each combination of values.

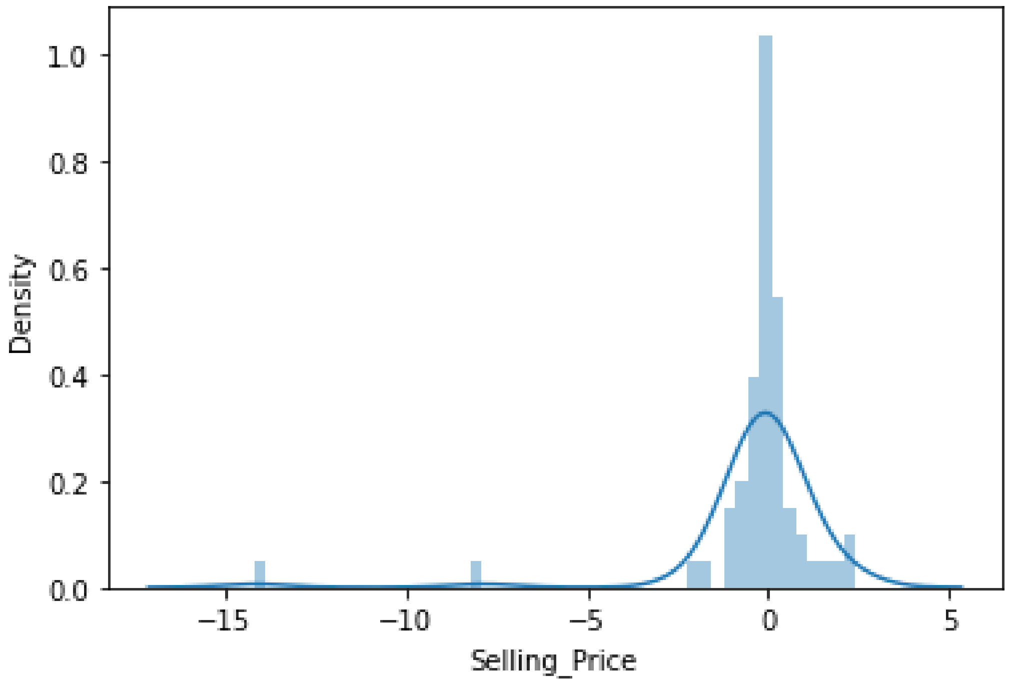
Cross Validation.

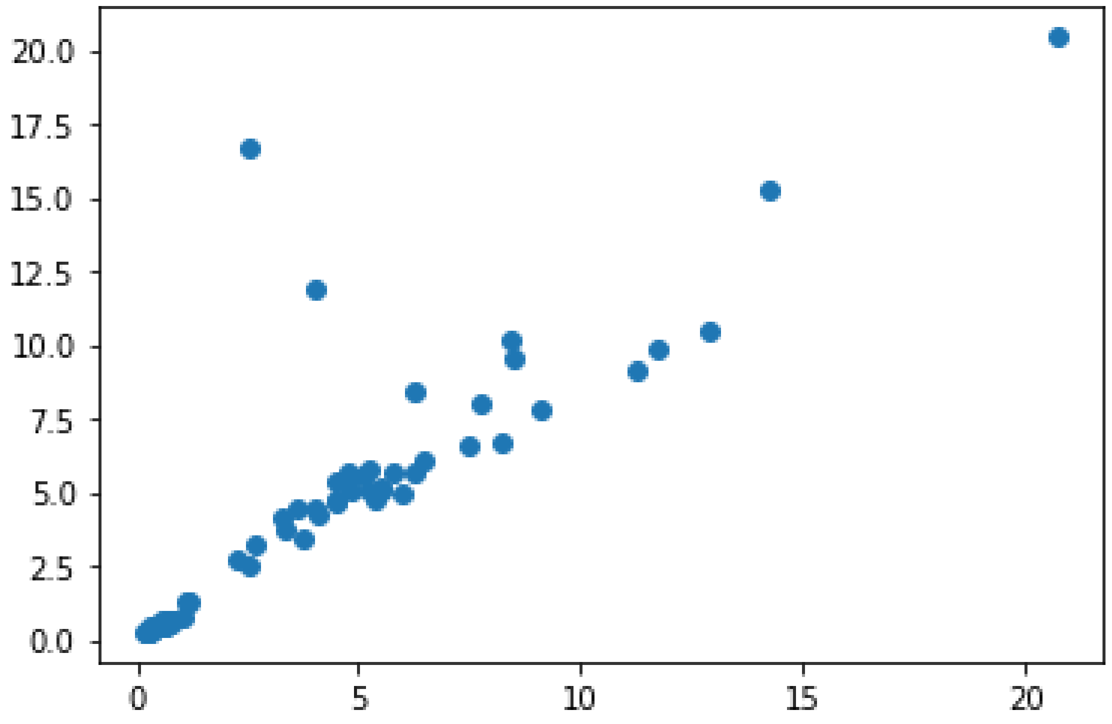
Results :

The predictions using our test data after RandomizedSearchCV and training the model using the training data. the predictions we are getting :

```
array([10.41779,  7.64262,  4.6179 ,  0.57719,  3.88675,  2.37943,
        8.0509 ,  1.11596,  0.76032,  5.1204 ,  6.32682, 11.66902,
        9.03532,  0.52968,  0.71901,  0.87698,  4.371  , 20.48701,
        1.074  ,  3.52876,  8.04853,  3.62571, 22.02013,  5.4299 ,
        5.17776,  3.66865,  0.51192,  0.45039,  3.85209,  6.74453,
        5.81529,  0.60187,  0.24235,  3.50707,  5.25594,  7.3224 ,
       11.18002,  6.86841,  5.4    ,  2.61038,  0.46557,  4.73289,
        7.51957,  9.32836,  0.57361,  5.86636,  2.70671,  0.75756,
        0.91893,  1.4024 ,  4.8493 ,  5.36836, 11.47684,  0.60498,
        2.85296,  3.06986,  0.31055,  0.81776,  5.72672,  0.563  ,
        3.53034])
```

Visualizing this by various plots :





This plot shows relation between out actual test data and the predicted value.

The vaious loss functions calculated :

1. MAE : Mean absolute error - It is the average of the sum of difference between the measured value and “true” value.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

2. MSE : Mean squared error - It is the average of the sum square of difference between the measured value and “true” value.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

3. RMSE : Root mean squared error - RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It’s the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Source : https://www.statisticshowto.com/absolute-error/

https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

Output of loss functions :

```
MAE: 0.82760213114754
MSE: 2.7328771084082053
RMSE: 1.65314158752606 7
```