House Price Prediction with Regression

Abstract

In this project house prices will be predicted given explanatory variables that cover many aspects of residential houses. Housing prices are an important reflection of the economy, and housing price ranges are of great interest for both buyers and sellers.Accurately estimating the value of real estate is an important problem for many stakeholders including house owners, house buyers, agents, creditors, and investors. After a preliminary study of the available algorithms and data review, it became apparent that the problem fell under the regression category. Thus, the study could focus on three feasible algorithms : Linear Regression, Decision Tree Regressor, Random Forest Regressor.We chose root-mean-square error (RMSE) for evaluation of these models' performance .The RMSE received by the Random Forest regressor model on training data lies between 2.6389 to 4.0935 which is less than other models.

1. Main Objectives:-

- Propose an efficient and valid algorithm for predicting the price of a house based on their different features.
- Comparing the efficiency and validity of different algorithms using various regression techniques : Linear Regression, Decision Tree Regressor, Random Forest Regressor.
- Report a comparative analysis on the different regression methods.
- Develop a real time predictive efficient algorithm for prediction of house price.

2. Status and Other Details:-

- Completed
- Percentage contribution of members:
  - Mangu Singh, B190641EE - 50%
  - Piyush Agarwal, B190889EE - 50%
- Total Spent time:
  - 4 Weeks
- Machine Learning Packages are used for in this Project:
  - Numpy, Pandas, Matplotlib, Scikit-learn

3. Major Stumbling Blocks:-

- The dataset used in this project comes from the UCI Machine Learning Repository. This data was collected in 1978 and each of the 506 entries represents aggregate information about 14 features of homes from various suburbs located in Boston.
- Predicting the price of a house based on their different features.

4 . Introduction:-

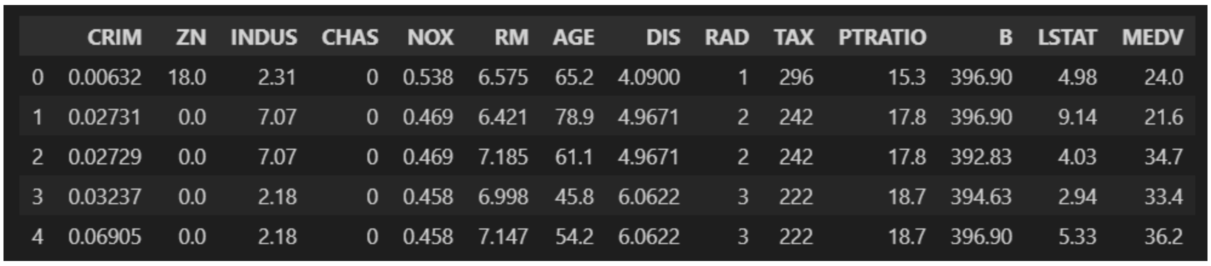
The sales price of a property can be predicted in various ways, but is often based on regression techniques. All regression techniques essentially involve one or more predictor variables as input and a single target variable as output.The problem that we are going to solve here is that given a set of features that describe a house in Boston, our machine learning model must predict the house price.we compare different machine learning methods performance in predicting the selling price of houses based on a number of features such as the per capita crime rate,, the number of average rooms and the geographical position etc.

5. Data Analysis:-

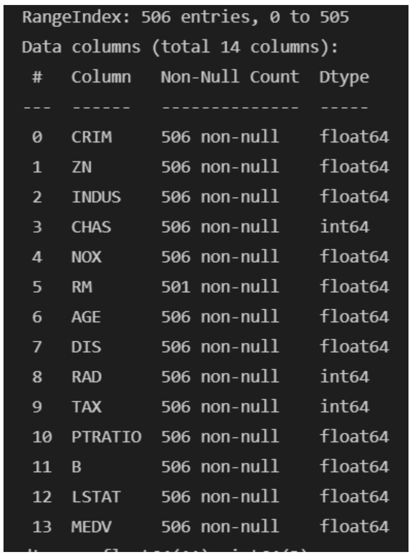
The dataset for this project originates from the UCI Machine Learning Repository. The Boston housing data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts.
For the purposes of this project, the following preprocessing steps have been made to the dataset:

I	CRIM	per capita crime rate by town
II	ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
III	INDUS	proportion of non-retail business acres per town
IV	CHAS	Charles River dummy variable (=1 if tract bounds river;0 otherwise)
V	NOX	nitric oxides concentration (parts per 10 million)
VI	RM	average number of rooms per dwelling
VII	AGE	proportion of owner-occupied units built prior to 1940
VIII	DIS	weighted distances to five Boston employment centres
IX	RAD	index of accessibility to radial highways
X	TAX	full-value property-tax rate per $10,000
XI	PTRATIO	pupil-teacher ratio by town
XII	B	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
XIII	LSTAT	% lower status of the population
XIV	MEDV	Median value of owner-occupied homes in $1000's

5.1 Sample Data:-

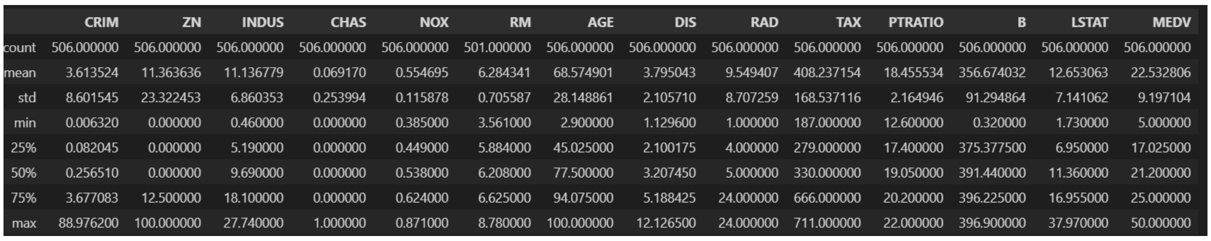


	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2



```
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       501 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
```

We have a MEDV(Price of the house) feature which is the target variable .So, We are going to add this to the main data frame.(independent variable).And also checking if there is any null value available in our data. We saw that there is no null value present .There are no null or missing values here.
Now we will analyse our data based on min,max and other parameters like mean,standard deviation and falling % category of particular data.

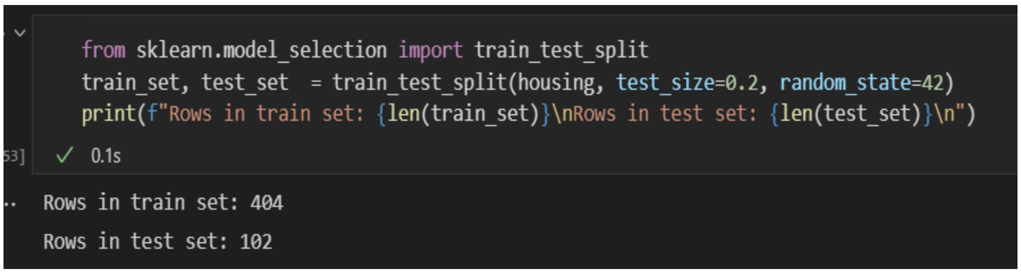


	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506.000000	506.000000	506.000000	506.000000	506.000000	501.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284341	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.705587	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.884000	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208000	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.625000	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

5.2 Train and Test data:-

To train our model in the best way , We are taking 80% of our Data for training purposes and 20% of our data for Testing purposes.

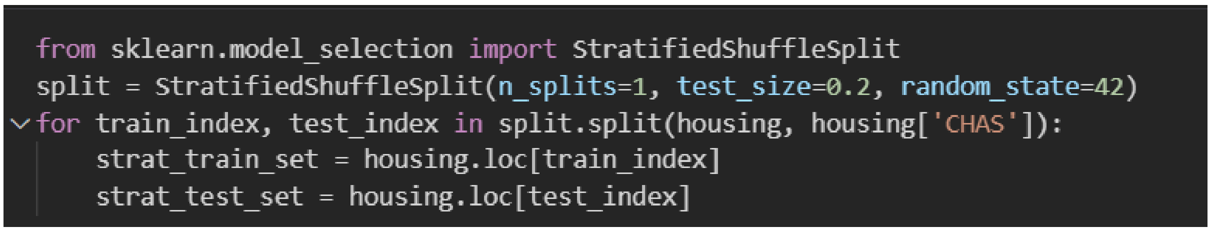
We have a total of 506 entries. It is useful to evaluate our model once it is trained. We want to know if it has learned properly from a training split of the data.



```
from sklearn.model_selection import train_test_split
train_set, test_set  = train_test_split(housing, test_size=0.2, random_state=42)
print(f"Rows in train set: {len(train_set)}\nRows in test set: {len(test_set)}\n")
```

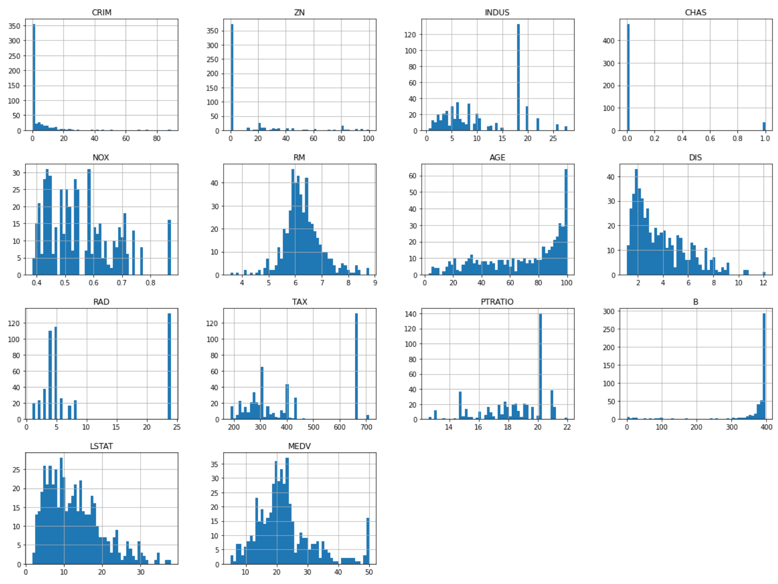
```
Rows in train set: 404
Rows in test set: 102
```

Here one important thing we can observe is that the value of **'CHAS'** variable is Zero in 75% cases. So, the important point is if we split our data in a way so that 1s' and 0s' of **'CHAS'** variable can also be divided into a ratio of 80/20 . Hence, Splitting will be better if We do it in the following way:-



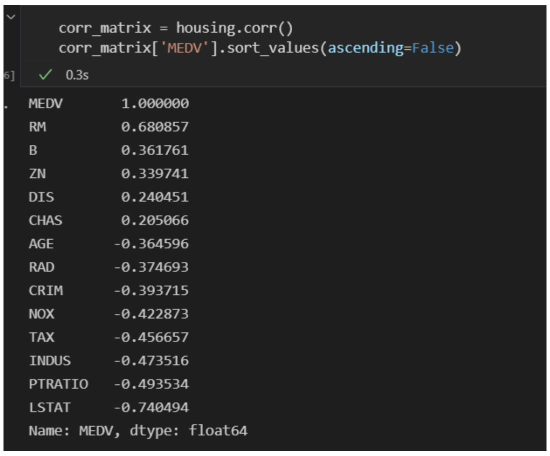
```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing['CHAS']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

5.3 Graphical Overview:-



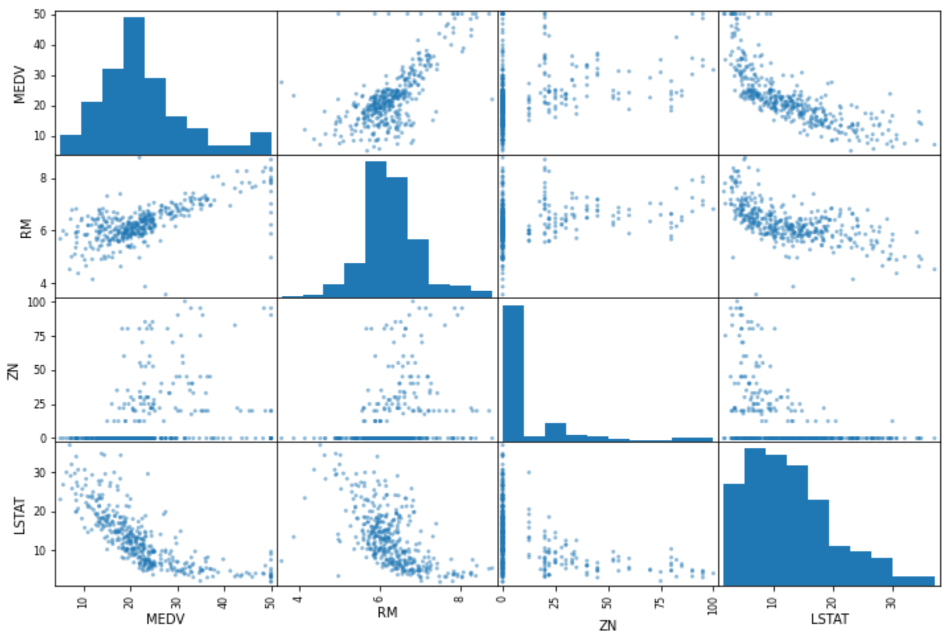
5.4 Feature Observations:-

Feature Selection is the process where we automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in our data can decrease the accuracy of the models and make our model learn based on irrelevant features.



```
corr_matrix = housing.corr()
corr_matrix['MEDV'].sort_values(ascending=False)
```

```
MEDV       1.000000
RM         0.680857
B          0.361761
ZN         0.339741
DIS        0.240451
CHAS       0.205066
AGE       -0.364596
RAD       -0.374693
CRIM      -0.393715
NOX       -0.422873
TAX       -0.456657
INDUS     -0.473516
PTRATIO   -0.493534
LSTAT     -0.740494
Name: MEDV, dtype: float64
```



6. Background Theory:-

6.1. Linear Regression:-

In our model the problem seems to be solved using linear regression because in our problem the expected outcome seems to be dependent on some features, and this type of problem is generally solved using linear regression. Linear regression is used to study the linear relationship between a dependent variable (y) and one or more independent variables (X). The linearity of the relationship between the dependent and independent variables is an *assumption* of the model. The relationship is modeled through a random disturbance term (or, error variable) ε. The disturbance is primarily important because we are not able to capture every possible influential factor on the dependent variable of the model. To capture all the other factors, not included as independent variables, that affect the dependent variable, the disturbance term is added to the linear regression model.

- In this way, the linear regression model takes the following form:

$$y = \mathbf{X}\beta + \varepsilon$$

$$= [\mathbf{1} \; x_1 \; x_2 \; \ldots \; x_K] \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_K \end{bmatrix} + \varepsilon$$

$$= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_K x_K + \varepsilon$$

- where

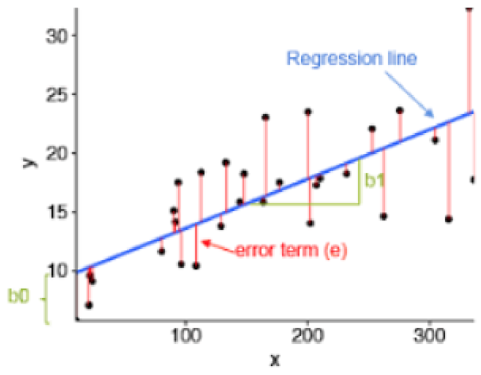
$$\beta_j \in \mathbb{R}, \quad j = 1, \ldots, K$$

- are the regression coefficients of the model (which we want to estimate!), and K is the number of independent variables included. The equation is called the regression equation.

6.2. Minimizing error:

"Minimize Error" in regression analysis is nothing but reducing the Residual distance such that the line fits close to the original points from the predicted points.

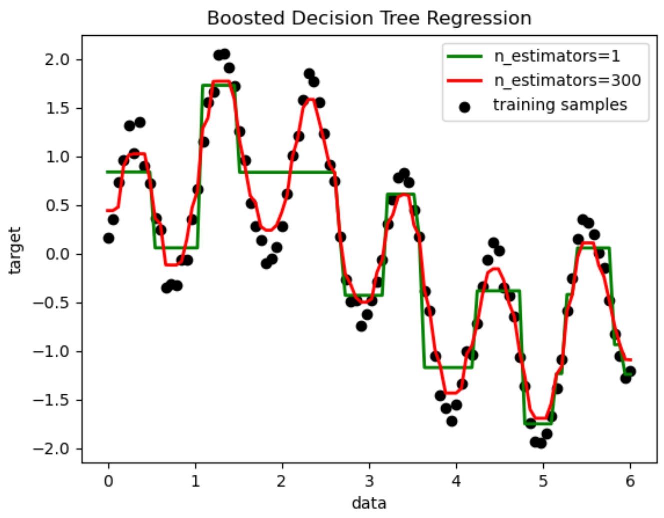
Note: The figure given below is for 2 - dimensional in the same way it can be in the more dimensions.



By using gradient descent algorithm and other algorithms we can minimize the mean square error.

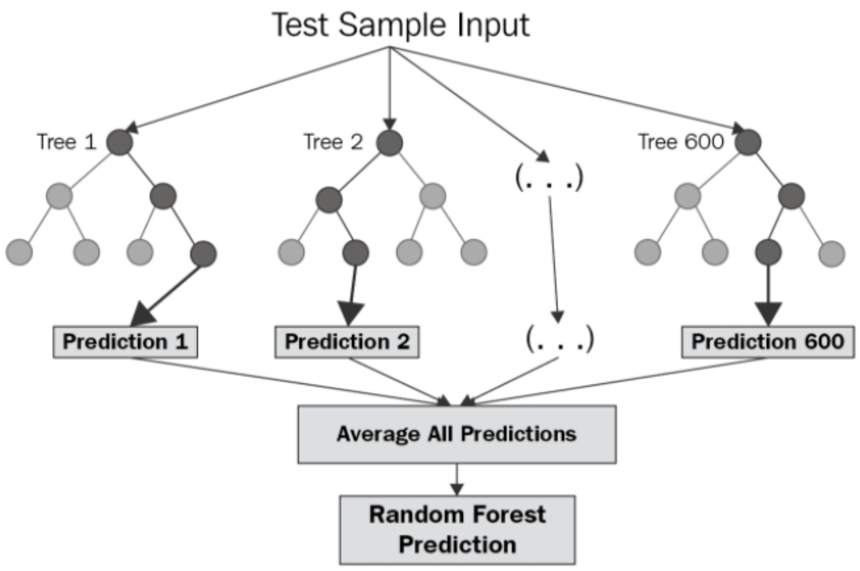
6.3. Decision Tree Regression:

Decision trees build 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 has two or more branches each representing values for the attribute tested. Leaf node 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.



6.4. Random Forest regressor:

Random forest is an algorithm which can be used both for classification and regression. Random forest models are constructed by using a collection of decision trees based on the training data. Instead of taking the target value from a single tree, the Random forest algorithm makes a prediction on the average prediction of a collection of trees. The decision trees themselves are constructed by fitting to randomly drawn groups of rows and columns in the training data. This method is called bagging, and results in a reduction of bias as each tree is built on different parts of the input at random. The method of averaging the predictions of decision trees reduces the overfitting that can occur when using single decision trees. The number of trees in the Random forest is an important hyperparameter of the algorithm called 'n_estimators' and the more trees used the more overfitting will be prevented. The tradeoff however, is an increase in the computation time needed. 'n_estimators' will be tested with different values in this study. Another hyperparameter of the Random forest algorithm is the 'criterion' hyperparameter which determines what error metric to use for measuring the quality of splits in the trees in the Random forest. The value can be either mean squared error or mean absolute error. A third hyperparameter that is particularly interesting for this study since there are a lot of features in the data set is 'max_features' which controls the number of features to consider when building the trees.



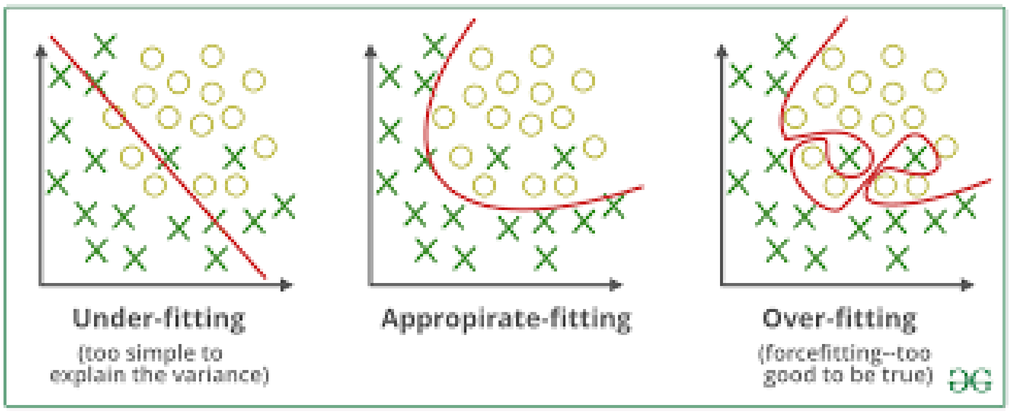
6.5. Root Mean Square Error:

The root-mean-square deviation (RMSD) or root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSD represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

6.6. Overfitting:

The programmer should know that there is a possibility that the output values may constitute an inherent noise which is the result of human or sensor errors. In this case, the algorithm must not attempt to infer the function that exactly matches all the data. Being too careful in fitting the data can cause overfitting, after which the model will answer perfectly for all training examples but will have a very high error for unseen samples. A practical way of preventing this is stopping the learning process prematurely, as well as applying filters to the data in the pre-learning phase to remove noises. Only after considering all these factors can we pick a supervised learning algorithm that works for the dataset we are working on. For example, if we were working with a dataset consisting of heterogeneous data, then decision trees would fare better than other algorithms. If the input space of the dataset we were working on had 1000 dimensions, then it's better to first perform PCA on the data before using a supervised learning algorithm on it.

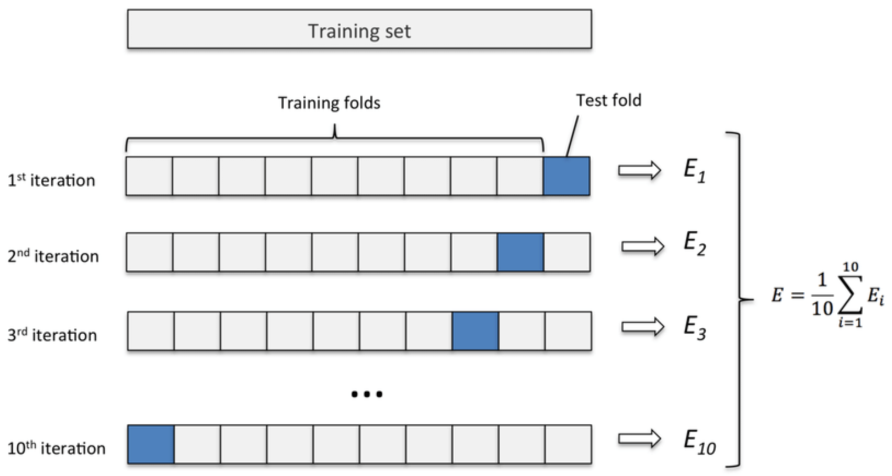

Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

6.7 Cross-validation :-

Cross-validation, sometimes called rotation estimation or out-of-sample testing, is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of *known data* on which training is run (*training dataset*), and a dataset of *unknown data* (or *first seen* data) against which the model is tested (called the validation dataset or *testing set*). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).

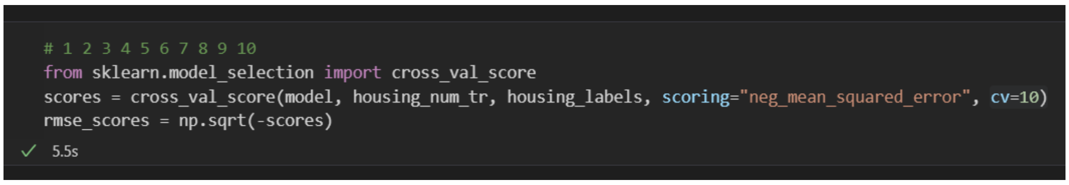
One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the *training set*), and validating the analysis on the other subset (called the *validation set* or *testing set*). To reduce variability, in most methods multiple rounds of cross-validation are performed using different partitions, and the validation results are combined (e.g. averaged) over the rounds to give an estimate of the model's predictive performance.

In summary, cross-validation combines (averages) measures of fitness in prediction to derive a more accurate estimate of model prediction performance.



7. **Model fitting**:-

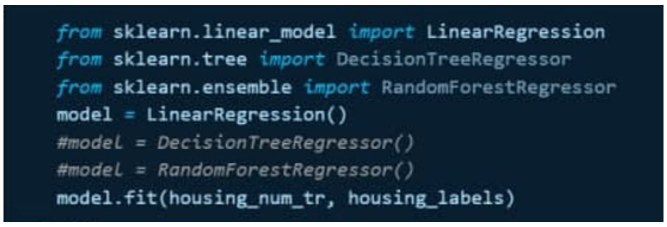
Comparing the efficiency and validity of different algorithms using the following three regression techniques : Linear Regression, Decision Tree Regressor, Random Forest Regressor. Here We are evaluating our models using cross validation techniques.



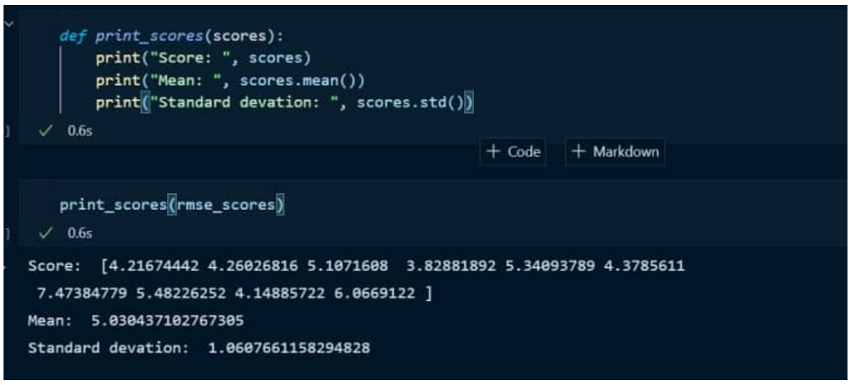
```
# 1 2 3 4 5 6 7 8 9 10
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, housing_num_tr, housing_labels, scoring="neg_mean_squared_error", cv=10)
rmse_scores = np.sqrt(-scores)
✓ 5.5s
```

7.1 Linear Regression Model Fitting:-

First of all we are using a Linear regression method to fit the curve. And we are getting the following results for RMSE on training data itself:-



```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
model = LinearRegression()
#model = DecisionTreeRegressor()
#model = RandomForestRegressor()
model.fit(housing_num_tr, housing_labels)
```



```python
def print_scores(scores):
    print("Score: ", scores)
    print("Mean: ", scores.mean())
    print("Standard devation: ", scores.std())
```

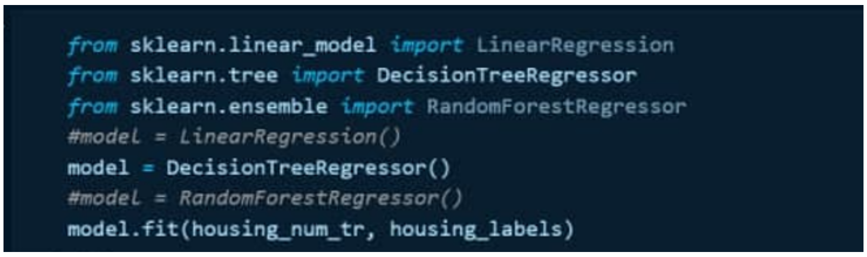
```python
print_scores(rmse_scores)
```

```
Score:  [4.21674442 4.26026816 5.1071608  3.82881892 5.34093789 4.3785611
 7.47384779 5.48226252 4.14885722 6.0669122 ]
Mean:  5.030437102767305
Standard devation:  1.0607661158294828
```

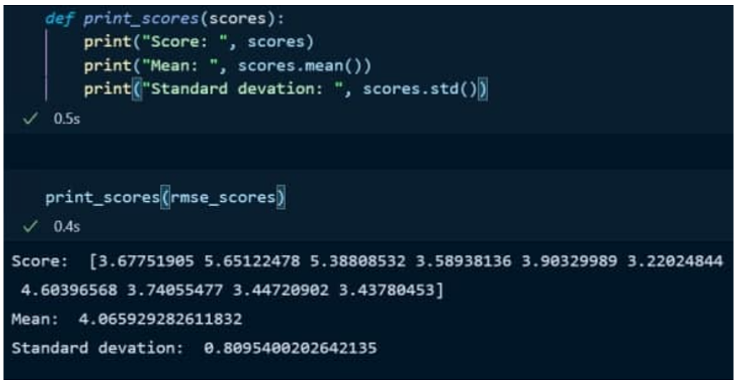
The RMSE value for Linear Regression model lies between 3.9696 to 6.0912.

7.2 Decision Tree Regression Model Fitting:-

we are getting the following results for RMSE on training data itself:-



```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
#model = LinearRegression()
model = DecisionTreeRegressor()
#model = RandomForestRegressor()
model.fit(housing_num_tr, housing_labels)
```



```python
def print_scores(scores):
    print("Score: ", scores)
    print("Mean: ", scores.mean())
    print("Standard devation: ", scores.std())
```

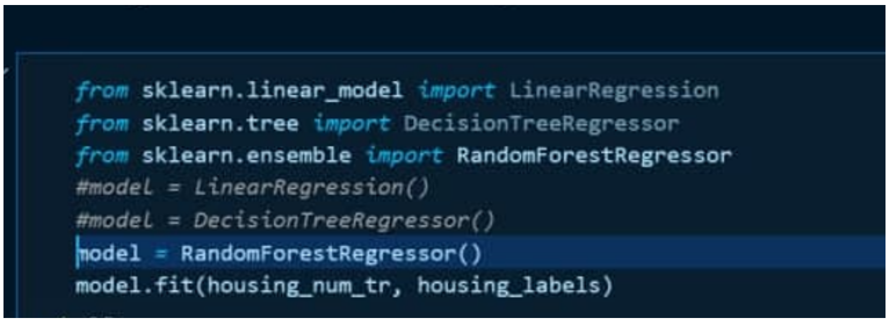
```python
print_scores(rmse_scores)
```

```
Score:  [3.67751905 5.65122478 5.38808532 3.58938136 3.90329989 3.22024844
 4.60396568 3.74055477 3.44720902 3.43780453]
Mean:  4.065929282611832
Standard devation:  0.8095400202642135
```

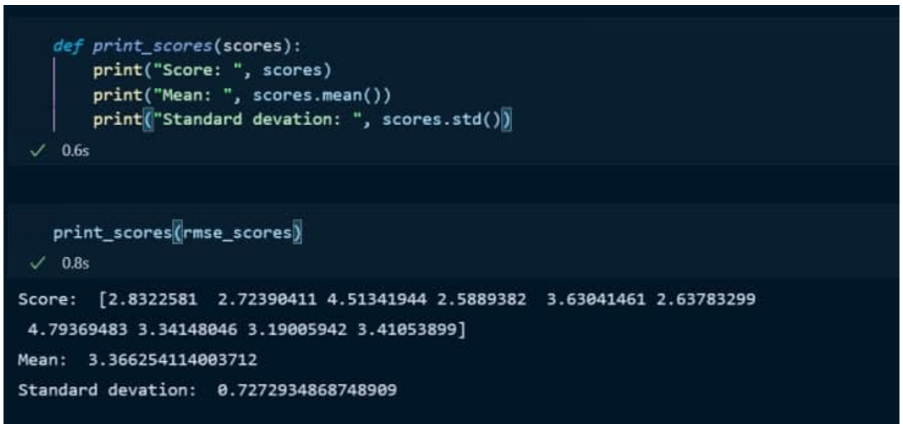
The RMSE value for Decision Tree regression model lies between 3.2563 to 4.8754

7.3 Random Forest Regressor Model Fitting:-

we are getting the following results for RMSE on training data itself:-



```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
#model = LinearRegression()
#model = DecisionTreeRegressor()
model = RandomForestRegressor()
model.fit(housing_num_tr, housing_labels)
```



```
def print_scores(scores):
    print("Score: ", scores)
    print("Mean: ", scores.mean())
    print("Standard devation: ", scores.std())
✓ 0.6s

print_scores(rmse_scores)
✓ 0.8s

Score:  [2.8322581  2.72390411 4.51341944 2.5889382  3.63041461 2.63783299
 4.79369483 3.34148046 3.19005942 3.41053899]
Mean:  3.366254114003712
Standard devation:  0.7272934868748909
```

The RMSE value for the Random Forest Regressor model lies between 2.6389 to 4.0935.

7.4 Values of mean and Deviation of RMSE :-

	Model name	Mean	Standard Deviation	RMSE
1	Linear Regression	5.030437103	1.060766116	3.9696 to 6.0912
2	Decision Tree	4.065929283	0.80954002	3.2563 to 4.8754
3	Random Forest Regressor	3.36625411	0.7272934869	2.6389 to 4.0935

8. Conclusion:-

The RMSE received by the Random Forest regressor model lies between 2.6389 to 4.0935 for training data which is less than other models and for the test data RMSE was 2.97 obtained. We have gone through how to implement the entire machine learning pipeline, and we have an intuitive understanding of machine learning algorithms.From the training Data Analysis, we could generate insight from the data. How each of the features relates to the target(MEDV). Also, it can be seen from the evaluation of three models that Random Forest Regressor performed better than Linear Regression and the Decision Tree Regression.