PROJECT REPORT

NATIONAL INSTITUTE OF TECHNOLOGY SIKKIM

ELECTRONICS AND COMMUNICATION ENGINEERING

- Course Name: **MACHINE LEARNING (CS16104)**
- Course Guide: **Dr. Hemant Kumar Kathania**

Presented BY : GROUP-5

- Vanka Janaki Rama Santhosh - B190044EC
- Andi Vivek - B190046EC

TOPIC:

Housing Prices Prediction Project Using Boston Data Set

- The dataset has house prices of the Boston residual areas. The expense of the house varies according to various factors like crime rate, number of rooms, etc.
  It is a good ML project for beginners to predict prices on the basis of new data.

Data Set:

https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

Introduction

In this project, we will develop and evaluate the performance and the predictive power of a model trained and tested on data collected from houses in Boston's suburbs. Once we get a good fit, we will use this model to predict the monetary value of a house located in Boston's area.

A model like this would be very valuable for a real estate agent who could make use of the information provided on a daily basis.

Getting the Data and Previous Preprocess

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.

The features can be summarized as follows:

- CRIM: This is the per capita crime rate by town
- ZN: This is the proportion of residential land zoned for lots larger than 25,000 sq.ft.
- INDUS: This is the proportion of non-retail business acres per town.

- CHAS: This is the Charles River dummy variable (this is equal to 1 if tract bounds river; 0 otherwise)
- NOX: This is the nitric oxides concentration (parts per 10 million)
- RM: This is the average number of rooms per dwelling
- AGE: This is the proportion of owner-occupied units built prior to 1940
- DIS: This is the weighted distances to five Boston employment centers
- RAD: This is the index of accessibility to radial highways
- TAX: This is the full-value property-tax rate per $10,000
- PTRATIO: This is the pupil-teacher ratio by town
- B: This is calculated as $1000(Bk — 0.63)^2$, where Bk is the proportion of people of African American descent by town
- LSTAT: This is the percentage lower status of the population
- MEDV: This is the median value of owner-occupied homes in $1000s

This is an overview of the original dataset, with its original features:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7

For the purpose of the project the dataset has been preprocessed as follows:

- The essential features for the project are: ‘RM’, ‘LSTAT’, ‘PTRATIO’ and ‘MEDV’. The remaining features have been excluded.
- 16 data points with a ‘MEDV’ value of 50.0 have been removed. As they likely contain censored or missing values.
- 1 data point with a ‘RM’ value of 8.78 it is considered an outlier and has been removed for the optimal performance of the model.
- As this data is out of date, the ‘MEDV’ value has been scaled multiplicatively to account for 35 years of markt inflation.

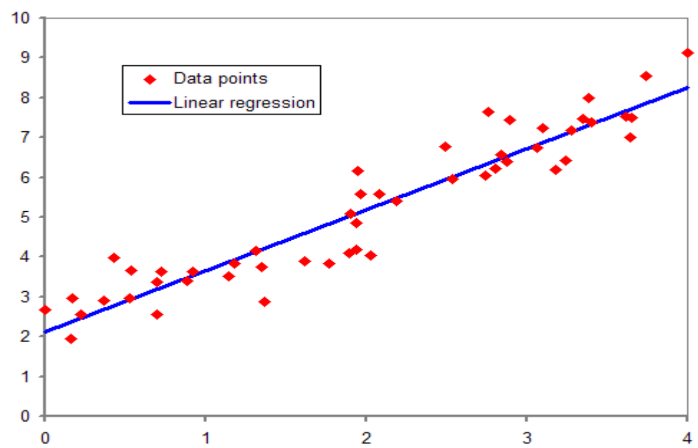
As our goal is to develop a model that has the capacity of predicting the value of houses, we will split the dataset into features and the target variable. And store them in features and prices variables, respectively

- The features ‘RM’, ‘LSTAT’ and ‘PTRATIO’, give us quantitative information abouth each datapoint. We will store them in features.
- The target variable, ‘MEDV’, will be the variable we seek to predict. We will store it at prices.

LINEAR REGRESSION

What is Linear Regression ?

Regression models are supervised learning models that are generally used when the value to be predicted is of discrete or quantitative nature. One of the most common examples where regression models are used is predicting the price of a house by training the data of sale of houses of that region.



The idea behind the Linear Regression model is to obtain a line that best fits the data. By best fit, what is meant is that the total distance of all points from our regression line should be minimal. Often this distance of the points from our regression line is referred to as an Error though it is technically not one. We know that the straight line equation is of the form:

$$y=mx+c$$

where y is the Dependent Variable, x is the Independent Variable, m is the Slope of the line and c is the Coefficient (or the y-intercept). Herein, y is regarded as the dependent variable as its value depends on the values of the independent variable and the other parameters.

This hypothesis maps our inputs to the output. The hypothesis for linear regression is usually presented as:

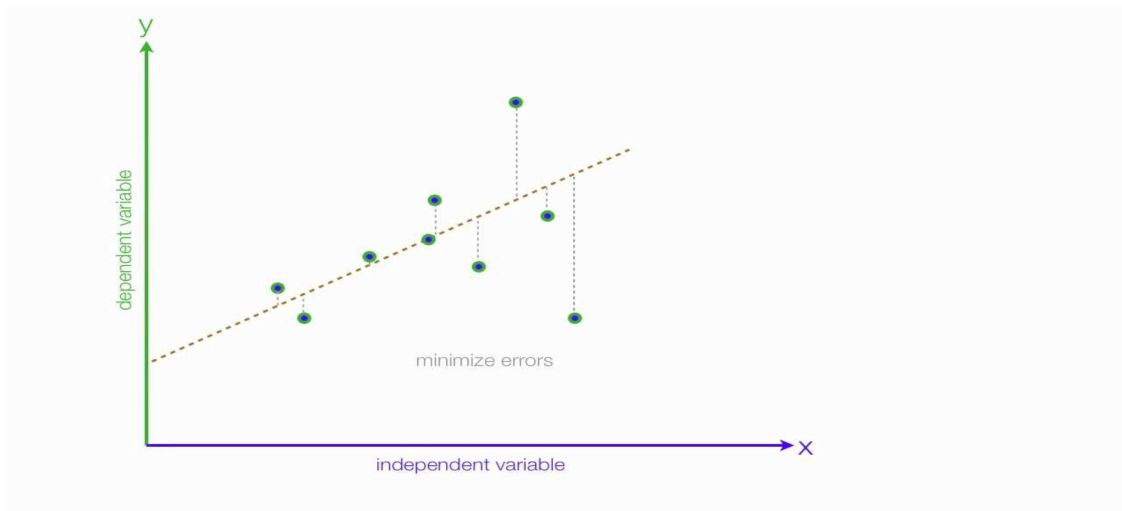
$$\boxed{h_\theta(x)= \theta_0 + \theta_1(x)}$$

Cost Function:

Cost functions are used to calculate how the model is performing. In layman's words, cost function is the sum of all the errors. While building our ML model, our aim is to **minimize** the cost function.

One common function that is often used in regression problems is the **Mean Squared Error** or **MSE**, which measure the difference between the known value and the predicted value.

$$\text{MSE}=\frac{1}{2m}\Sigma\left(h_\theta(\mathrm{x})^{(i)}-y^i\right)^2$$



ALGORITHM:

In [6]:
```python
# Importing the Libraries
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [7]:
```python
# Importing the Boston Housing dataset
from sklearn.datasets import load_boston
boston = load_boston()
```

In [8]:
```python
# Initializing the dataframe
data = pd.DataFrame(boston.data)
```

In [9]:
```python
# See head of the dataset
data.head()
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [10]:
```python
#Adding the feature names to the dataframe
data.columns = boston.feature_names
data.head()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [11]:
```python
#Adding target variable to dataframe
data['PRICE'] = boston.target
# Median value of owner-occupied homes in $1000s
```

In [12]:
```python
data.shape
```

Out[12]:
```
(506, 14)
```

In [13]:
```python
data.columns
```

Out[13]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'PRICE'],
      dtype='object')
```

In [14]:
```
data.dtypes
```

Out[14]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
PRICE      float64
dtype: object
```

In [15]:
```
data.nunique()
```

Out[15]:
```
CRIM       504
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        356
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      455
PRICE      229
dtype: int64
```

In [16]:
```
# Check for missing values
data.isnull().sum()
```

Out[16]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

In [17]:
```
# See rows with missing values
data[data.isnull().any(axis=1)]
```

Out[17]:

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE

In [18]:
```
# Viewing the data statistics
data.describe()
```

Out[18]:

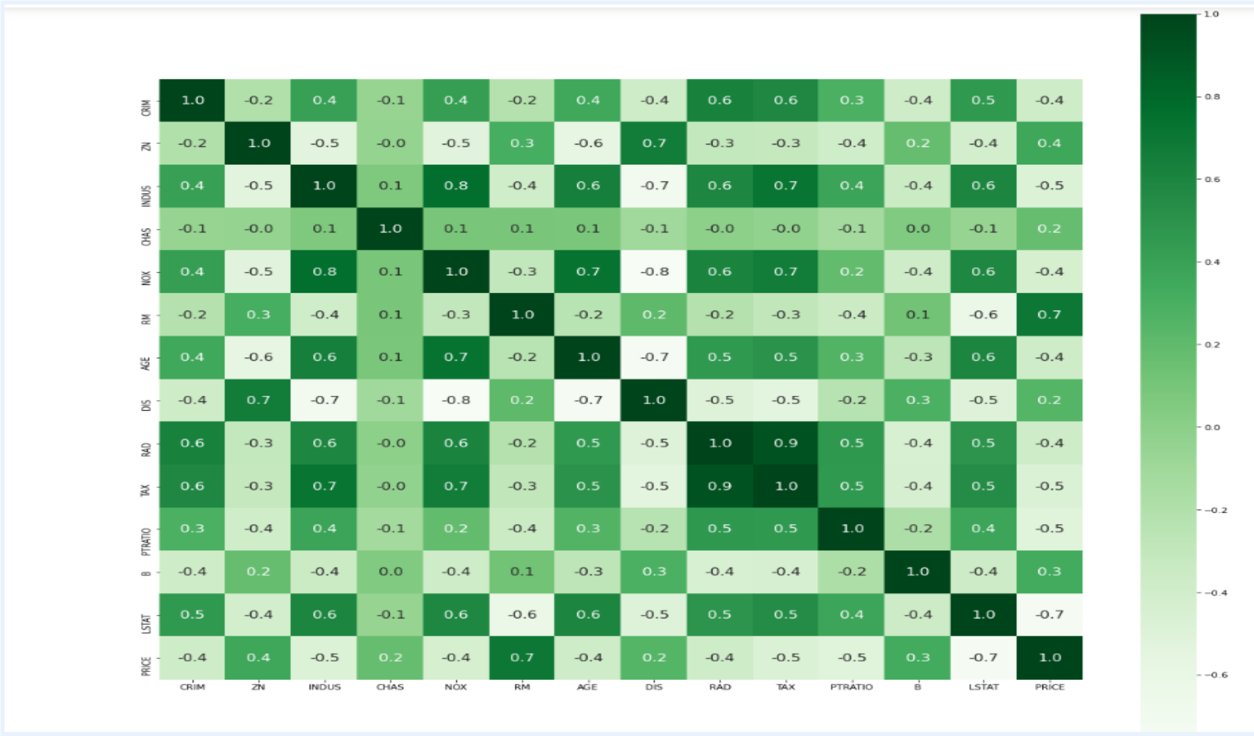
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.65
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.14
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	1.73
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.95
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.36
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.95
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	37.97

In [19]:
```
# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

Out[19]: (14, 14)

In [20]:
```
# Plotting the heatmap of correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'size':15}, cmap='Greens')
```

Out[20]: <AxesSubplot:>




```
In [21]: # Spliting target variable and independent variables
         X = data.drop(['PRICE'], axis = 1)
         y = data['PRICE']
```

```
In [22]: # Splitting to training and testing data

         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 4)
```

```
In [23]: # Import library for Linear Regression
         from sklearn.linear_model import LinearRegression

         # Create a Linear regressor
         lm = LinearRegression()

         # Train the model using the training sets
         lm.fit(X_train, y_train)

Out[23]: LinearRegression()
```

```
In [24]: # Value of y intercept
         lm.intercept_

Out[24]: 36.357041376595284
```

```
In [25]: #Converting the coefficient values to a dataframe
         coeffcients = pd.DataFrame([X_train.columns,lm.coef_]).T
         coeffcients = coeffcients.rename(columns={0: 'Attribute', 1: 'Coefficients'})
         coeffcients
```

Out[25]:

	Attribute	Coefficients
0	CRIM	-0.12257
1	ZN	0.0556777
2	INDUS	-0.00883428
3	CHAS	4.69345
4	NOX	-14.4358
5	RM	3.28008
6	AGE	-0.00344778
7	DIS	-1.55214
8	RAD	0.32625
9	TAX	-0.0140666
10	PTRATIO	-0.803275
11	B	0.00935369
12	LSTAT	-0.523478

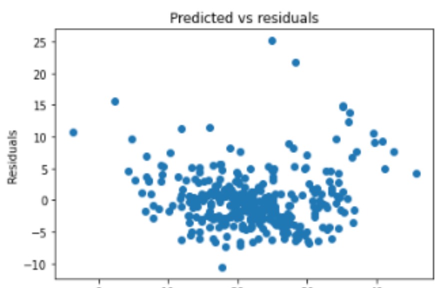
```
In [26]: # Model prediction on train data
         y_pred = lm.predict(X_train)
```

```
In [27]: # Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred)) #Mean Absolute Error
print('MSE:',metrics.mean_squared_error(y_train, y_pred))    #Mean Square Erroe
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))    #Root Mean Square Error
```

```
R^2: 0.7465991966746854
Adjusted R^2: 0.736910342429894
MAE: 3.089861094971132
MSE: 19.073688703469035
RMSE: 4.367343437774162
```

```
In [28]: # Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```


```
In [29]: plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```




```
In [30]: # Predicting Test data with the model
y_test_pred = lm.predict(X_test)
```

```
In [31]: # Model Evaluation
acc_linreg = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_linreg)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.7121818377409187
Adjusted R^2: 0.6850685326005705
MAE: 3.859005592370742
MSE: 30.053993307124216
RMSE: 5.482152251362982
```

Decision Tree Regression

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Discrete output example: A weather prediction model that predicts whether or not there'll be rain on a particular day.

Continuous output example: A profit prediction model that states the probable profit that can be generated from the sale of a product.

We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.

How is Splitting Decided for Decision Trees?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.Decision trees regression normally use mean squared error (MSE) to decide to split a node in two or more sub-nodes.
Suppose we are doing a binary tree the algorithm first will pick a value, and split the data into two subset. For each subset, it will calculate the MSE separately. The tree chooses the value which results in the smallest MSE value.

Let's examine how Splitting Decided for Decision Trees Regressor in more detail. The first step to create a tree is to create the first binary decision. How are you going to do it?

- We need to pick a variable and the value to split on such that the two groups are as different from each other as possible.
- For each variable, for each possible value of the possible value of that

variable see whether it is better.
- How to determine if it is better? Take weighted average of two new nodes (mse*num_samples)

To sum up, we now have:

- A single number that represents how good a split is is the weighted average of the mean squared errors of the two groups that create.
- A way to find the best split is to try every variable and to try every possible value of that variable and see which variable and which value gives us a split with the best score.

This is the entirety of creating a decision tree regressor and will stop when some stopping condition (defined by hyperparameters) is met:

- When you hit a limit that was requested (for examplemax_depth)
- When your leaf nodes only have one thing in them (no further split is possible, MSE for the train will be zero but will overfit for any other set -not a useful model)

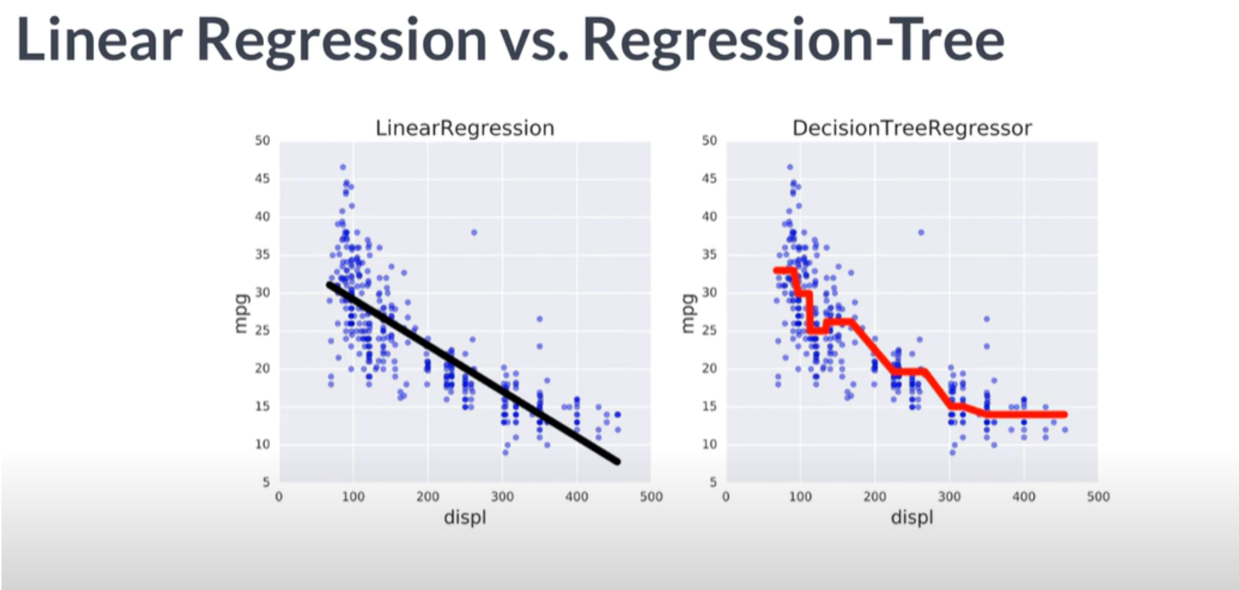
Information Criterion for Regression-Tree

$$I(\text{node}) = \underbrace{\text{MSE(node)}}_{mean-squared-error} = \frac{1}{N_{node}} \sum_{i \in node} \left(y^{(i)} - \hat{y}_{node}\right)^2$$

$$\underbrace{\hat{y}_{node}}_{mean-target-value} = \frac{1}{N_{node}} \sum_{i \in node} y^{(i)}$$

The MSE at each node is calculated for which underlying model?

The underlying model is simply the average of the data points. For the initial root mode is what if we just predicted the average of the dependent variable of all our training data points. Another possible option would be instead of using the average to use the median or we can even run a linear regression model. There are a lot of things we could do but in practice the average works really well.They do exist random forest models where the leaf nodes are independent linear regressions but they're not widely used.



ALGORITHM:

```
In [1]: import pandas as pd
```

```
In [2]: column_names = ['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT','MEDV']
        df = pd.read_csv('C:\\Users\\vjrs6\\Downloads\\housing.csv',delimiter=r"\s+",names=column_names)
```

```
In [3]: df.head()
```

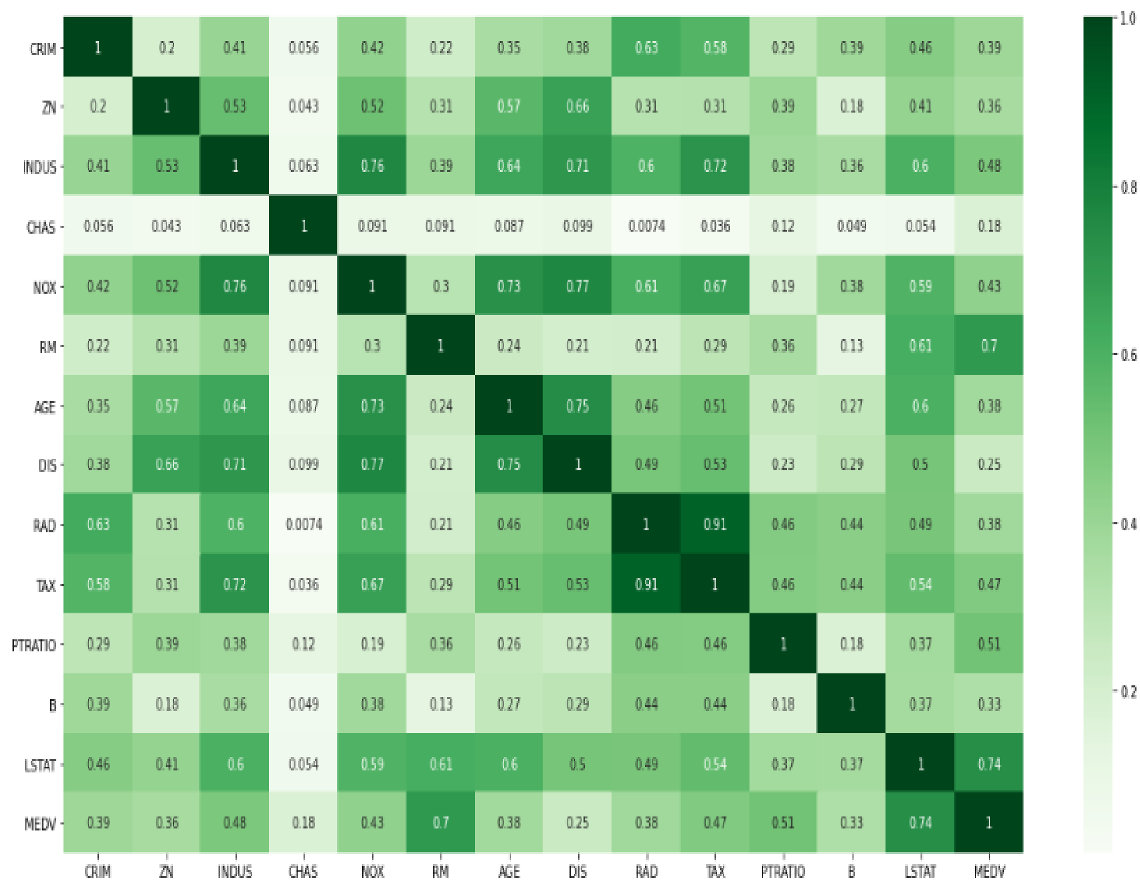
Out[3]:

	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	296.0	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	242.0	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	242.0	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	222.0	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	222.0	18.7	396.90	5.33	36.2

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [5]:
```
corr_m = df.corr()
import seaborn as sn
import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
sn.heatmap(corr_m.abs(),annot=True,cmap='Greens')
plt.show()
```



```
In [6]: X_data = df.drop('MEDV',axis=1)
        Y_data = df['MEDV']

In [7]: from sklearn.model_selection import train_test_split

In [8]: x_train, x_test, y_train, y_test = train_test_split(X_data,Y_data,train_size=0.8,random_state=0)

In [17]: from sklearn.tree import DecisionTreeRegressor
         model = DecisionTreeRegressor()

In [18]: model.fit(x_train,y_train)

Out[18]: DecisionTreeRegressor()

In [11]: model.score(x_train,y_train)*100

Out[11]: 100.0

In [12]: y_pred = model.predict(x_train)

In [13]: from sklearn import metrics

In [14]: print('MSE:',metrics.mean_absolute_error(y_train,y_pred))
         print('MAE:',metrics.mean_absolute_error(y_train,y_pred))
         print('RMSE:',metrics.mean_squared_error(y_train,y_pred))

         MSE: 0.0
         MAE: 0.0
         RMSE: 0.0

In [15]: y_predict = model.predict(x_test)

In [16]: model.score(x_test,y_predict)*100

Out[16]: 100.0

In [19]: print('MSE:',metrics.mean_absolute_error(y_test,y_predict))
         print('MAE:',metrics.mean_absolute_error(y_test,y_predict))
         print('RMSE:',metrics.mean_squared_error(y_test,y_predict))

         MSE: 3.547058823529411
         MAE: 3.547058823529411
         RMSE: 31.516078431372545

In [ ]:
```

Random Forest Algorithm

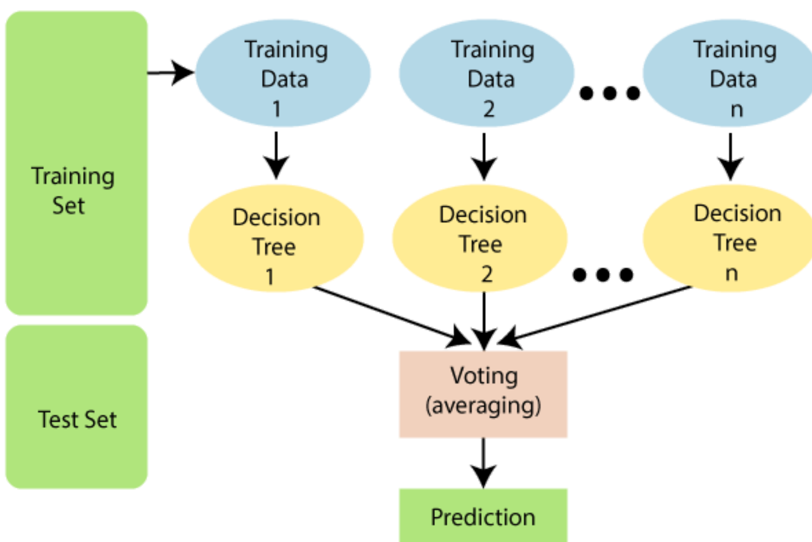
Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and

Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



How does the Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision trees, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Alogthrim:-

In [14]:
```
# Viewing the data statistics
data.describe()
```

Out[14]:

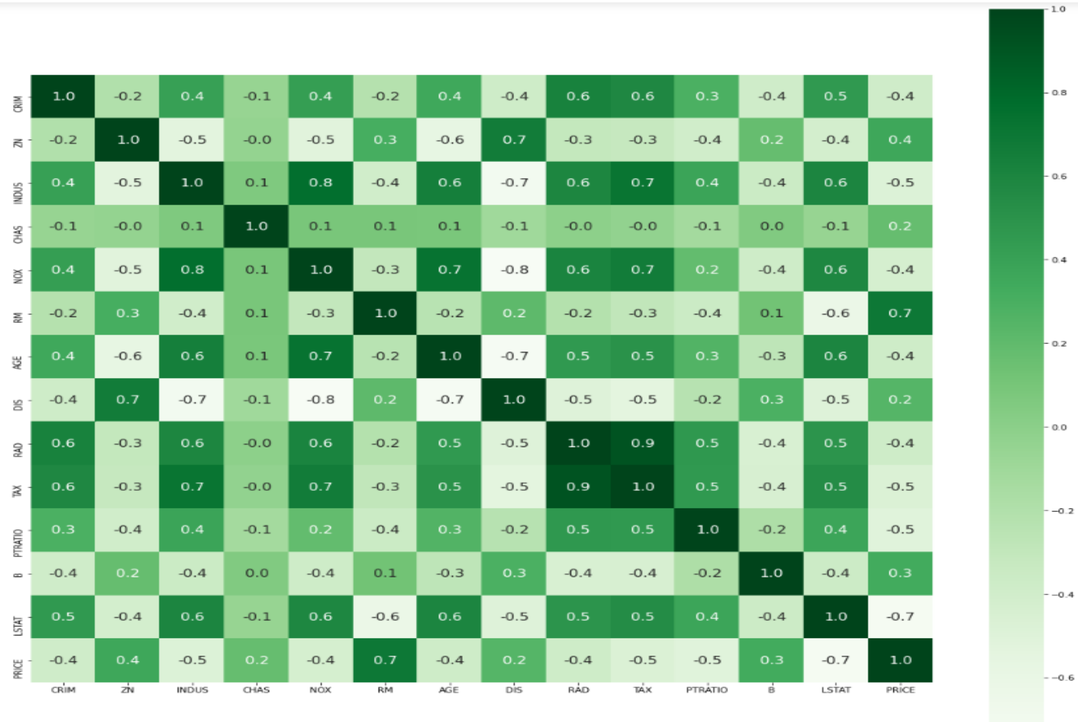
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0(
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.6!
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.14
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	1.73
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.9!
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3(
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.9!
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	37.9;

In [15]:
```
# Finding out the correlation between the features
corr = data.corr()
corr.shape
```

Out[15]: (14, 14)

In [16]:
```
# Plotting the heatmap of correlation between features
plt.figure(figsize=(20,20))
sns.heatmap(corr, cbar=True, square= True, fmt='.1f', annot=True, annot_kws={'size':15}, cmap='Greens')
```

Out[16]: <AxesSubplot:>



In [17]:
```
# Spliting target variable and independent variables
X = data.drop(['PRICE'], axis = 1)
y = data['PRICE'] #dependent variable
```

In [18]:
```
# Splitting to training and testing data

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 4)
```

In [19]:
```
# Import Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regressor
reg = RandomForestRegressor()

# Train the model using the training sets
reg.fit(X_train, y_train)
```

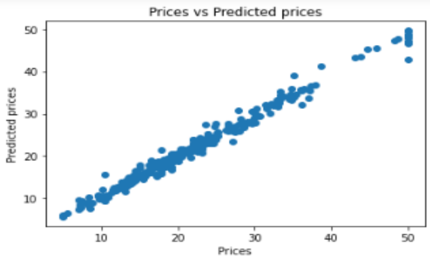
Out[19]: RandomForestRegressor()

In [20]:
```
# Model prediction on train data
y_pred = reg.predict(X_train)
```

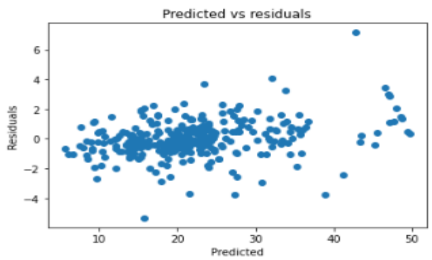
In [21]:
```
# Model Evaluation
print('R^2:',metrics.r2_score(y_train, y_pred))
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_train, y_pred))*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_train, y_pred))
print('MSE:',metrics.mean_squared_error(y_train, y_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
```

```
R^2: 0.981709310361639
Adjusted R^2: 0.9810099604637017
MAE: 0.8156610169491523
MSE: 1.3767553841807934
RMSE: 1.1733521995465783
```

In [22]:
```
# Visualizing the differences between actual prices and predicted values
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```



In [23]:
```
# Checking residuals
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



In [24]:
```
# Predicting Test data with the model
y_test_pred = reg.predict(X_test)
```

In [25]:
```
# Model Evaluation
acc_rf = metrics.r2_score(y_test, y_test_pred)
print('R^2:', acc_rf)
print('Adjusted R^2:',1 - (1-metrics.r2_score(y_test, y_test_pred))*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print('MAE:',metrics.mean_absolute_error(y_test, y_test_pred))
print('MSE:',metrics.mean_squared_error(y_test, y_test_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))
```

```
R^2: 0.8291166522261035
Adjusted R^2: 0.8130189455517509
MAE: 2.4945921052631586
MSE: 17.843651526315796
RMSE: 4.2241746562276274
```

CONCLUSION:-

BY doing all algorithm using same data set we conclude that the Accuracies are different for three and the decision Tree Algorithm having good accuracy and we use to prefer this algorithm to improve the model for further

Accuracy:

	Linear Regression	Random Forest	Decision Tree
Training :	74%	98%	100%
Testing:	71%	82%	100%

Preferences:-

- James, Gareth, et al. "Tree-based methods." An introduction to statistical learning. Springer, New York, NY, 2021. 327-365.
- Kaggle
- Gdcoder.com
- towardsdatascience.com
- Javatpoint.com

THANK YOU