Boosting

Ada Boost (Adaptive Boosting)

For understanding this algorithm we will use the following simple dataset for heart patient predicton

In [3]:

```
import pandas as pd
heart_data = pd.read_csv('https://raw.githubusercontent.com/training-ml/Files/main/heart_di
heart_data
```

Out[3]:

	Unnamed: 0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
0	0	63	1	3	145	233	1	0	150	0	2.3	0	0
1	1	37	1	2	130	250	0	1	187	0	3.5	0	0
2	2	41	0	1	130	204	0	0	172	0	1.4	2	0
3	3	56	1	1	120	236	0	1	178	0	0.8	2	0
4	4	57	0	0	120	354	0	1	163	1	0.6	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	298	57	0	0	140	241	0	1	123	1	0.2	1	0
299	299	45	1	3	110	264	0	1	132	0	1.2	1	0
300	300	68	1	0	144	193	1	1	141	0	3.4	1	2
301	301	57	1	0	130	131	0	1	115	1	1.2	1	
302	302	57	0	1	130	236	0	0	174	0	0.0	1	

303 rows × 15 columns

Python Implementation

In [5]:

```
# 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

import warnings
warnings.filterwarnings('ignore')
```

In [6]:

```
# Initializing the dataframe
data=pd.read_csv('https://raw.githubusercontent.com/training-ml/Files/main/boston_house_ren
```

In [7]:

```
# see head of the dataset
data.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.

In [8]:

```
#Any missing values
data.isnull().sum()
```

Out[8]:

```
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 [9]:

```
# Dataset analysis/stats using describe method.
data.describe()
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	50
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	1

In [10]:

```
# Finding out the correlation between the features

corr=data.corr()
corr.shape
```

Out[10]:

```
(14, 14)
```

In [11]:

```
corr
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996
PRICE	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929

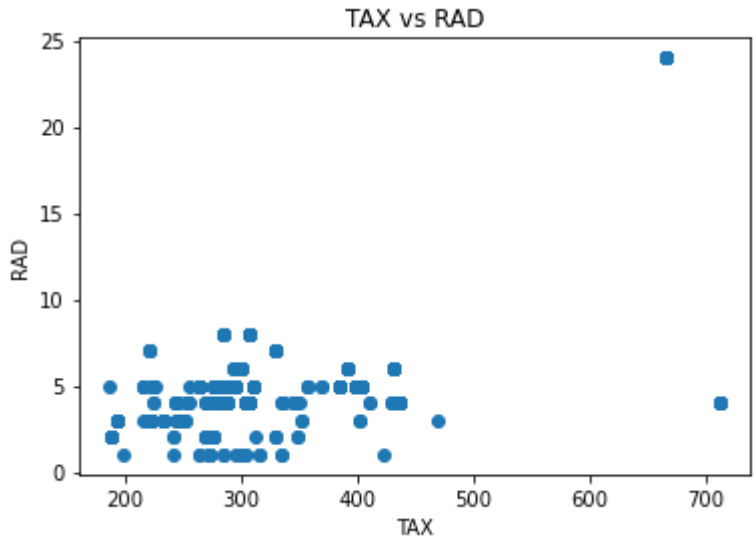
In [12]:

```
# plotting the heatmap of correlation between features
plt.figure(figsize=(12,10))
sns.heatmap(corr, annot=True)
plt.show()
```

Seems RAD and TAX are correlated. Lets check with scatter plot

In [14]:

```
plt.scatter(data.TAX,data.RAD)
plt.xlabel('TAX')
plt.ylabel('RAD')
plt.title('TAX vs RAD')
plt.show()
```



Since we cannot confirm with plot that , there is a close relation. So we will not take risk and not deleting any features.

In [15]:

```
# dividing feature and label data
x=data.drop(columns=['PRICE'], axis=1)
y= data['PRICE']
```

In [16]:

```
# train test split (hold out method)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=100)
```

AdaBoost model training

In [17]:

```
from sklearn.ensemble import AdaBoostRegressor
```

In [18]:

```
ada= AdaBoostRegressor()
```

In [19]:

```
ada.fit(x_train,y_train)
```

Out[19]:

```
AdaBoostRegressor()
```

In [20]:

```
#Model prediction on train data
y_pred=ada.predict(x_train)
```

In [21]:

```
accuracy=metrics.r2_score(y_train,y_pred)
print('R square score', accuracy)
```

```
R square score 0.9180426801956976
```

Since model has already seen the data, so it is producing better score.

In [22]:

```
# predicting Test data with the model
y_test_pred=ada.predict(x_test)
```

In [23]:

```
# Model Evaluation
accuracy=metrics.r2_score(y_test,y_test_pred)
print('R square score', accuracy)
```

```
R square score 0.8093034208894052
```

Hyperparameter Tuning using RandomizedSearchCV

In [24]:

```
from sklearn.model_selection import RandomizedSearchCV
```

In [25]:

```
params={'n_estimators':[47,50,60,70],'learning_rate':[0.25,0.30,0.40]}
```

In [28]:

```
rnd_srch=RandomizedSearchCV(AdaBoostRegressor(),cv=5,param_distributions=params)
```

In [29]:

```
rnd_srch.fit(x_train,y_train)
```

Out[29]:

```
RandomizedSearchCV(cv=5, estimator=AdaBoostRegressor(),
                   param_distributions={'learning_rate': [0.25, 0.3, 0.4],
                                        'n_estimators': [47, 50, 60, 70]})
```

In [31]:

```
rnd_srch.best_estimator_
```

Out[31]:

```
AdaBoostRegressor(learning_rate=0.4, n_estimators=60)
```

In [32]:

```
ada=AdaBoostRegressor(learning_rate=0.4,n_estimators=60)
ada.fit(x_train,y_train)
y_pred=ada.predict(x_test)
print('********  accuracy post tuning**********')
print(metrics.r2_score(y_test,y_pred))
```

```
********  accuracy post tuning**********
0.8389164273924564
```

In [ ]:

```

```