Consider the Boston housing prices data set.  Inspect it, and create one record (one row) and show it as the first part of your answer.  Leave the predicted variable Y\_i as unknown (property price in 1000$). In this context, answer the below questions integrating your answers like an essay (tell a story):

[Boston Housing Price Data (Links to an external site.)](https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html)

1) What machine learning algorithms would you use to predict the property price (home price Y\_i)?

2) Suppose we convert the home Price into a multiclass category.  how would you achieve that using binning (converting numbers to classes)?  what algorithms would you use now?

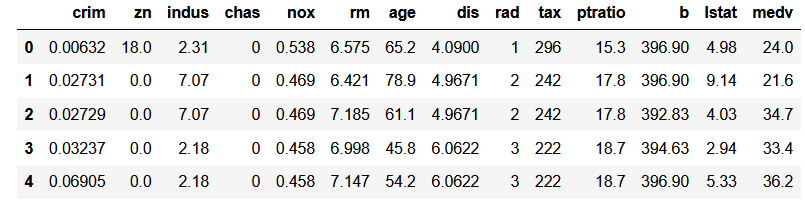
Specifically in the context of predicting the answers for this one record, please answer the below questions:

3) What is bagging and how does it work?

4) How does boosting work, and is it better than bagging? Why?

5) Briefly explain what these algorithms are and how they work: AdaBoost and XGBoost

The Boston Housing Prices data set contains 506 rows (records) with 14 columns (features).



**Fig.1 First 5 rows in the dataset.**

The target variable for the noted dataset is “medv” which stands for **Median value of owner-occupied homes in $1000's.**

Chart, treemap chart

Description automatically generated

**Fig.2 Correlation plot between variables in dataset.**

For this particular problem when we want to predict a price (continuous variables) based on multiple features a multiple regression is the best way to solve this problem. The regression itself can be done by multiple algorithms such as: multiclass linear regression, regression trees, regression assemble models (Random Forest for Regression).

Another way to solve this prediction problem could be by converting the “medv” target variable into a multi-category variable by splitting the data in different buckets (Ex: High, Medium, Low) based on the distribution of such variables and some desired boundaries to establish the side of the bucket (category). By doing this the problem now can be solve by a classification model, in this case depending on the data and the use of the model some models that can be used are logistic regression, decision trees, random forest classification or another ensemble models. For this example, I would compare different ensemble models (bagging & boosting) to see which one has better performance (accuracy & AUC) on this dataset.

Bagging (also called aggregation or bootstrapping) which could be an option here is a general method that enhances the performance of some of the models by construing n number of models by bootstrapping the training dataset and then in the case of regression obtaining the average of the predictions of each model or in the case of classification take a majority vote to decide the final prediction from the bagging model.

Another approach to improve the performance on a single model (decision tree) is boosting. Boosting in general is like bagging with the main difference being that the trees are grown sequentially and not independent of the other trees as in the case of bagging. In the case of boosting each tree is the modified version of the previous one. This method/approach learns slowly as it is improving slightly on each iteration (this also is related to one of its hyperparameters: the shrinkage parameter λ). As a side note usually statistical learning methods that learn slowly tend to perform better, for this reason usually boosting tends to perform better than bagging.

Chart, bubble chart

Description automatically generated

**Fig.3 Bagging Vs Boosting (Julia Nikulski. “The Ultimate Guide to AdaBoost, random forest and XGBoost” by Nikulski, Mar 16, 2020. Towards Data Science,** [**https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f**](https://towardsdatascience.com/the-ultimate-guide-to-adaboost-random-forests-and-xgboost-7f9327061c4f) **)**

Two of the most popular boosting methods for decision trees are: AdaBoost and XGBoost.

**AdaBoost works in the following way:**

It is a sequential series of decision trees as weak learners that punishes wrong predictions by assigning bigger weights to those after each training round. The main purpose of the noted assignation of bigger weights is to give the model the ability to learn from previous mistakes. The final prediction (for classification process) is the weighed majority vote. The general idea of the algorithm is as follows:

* Calculate the distribution p by normalizing the weight vector w (initially w is 1/N , where N is the number of rows or data points).
* Grow a weak learner (for decision trees these are called stumps) using the previous distribution p. Return the prediction for each example).
* Calculate the error e between the prediction and the actual label.
* Assign a weight rate based on the error e, β = e/(1-e).
* Update the weight vector to w = w\* β. This to allow predictions with poor performance to have higher weights and allow the algorithm to learn from its mistakes.

**XGBoost works in the following way:**

XGBoost (eXtreme Gradient Boosting) is a relatively new algorithm, developed to increase speed and performance while providing a way to use regularization to reduce overfitting.

Similarly, as AdaBoost and all boosting ensemble methods, XGBoost is a sequential learning process. Gradient descent is used to compute the optimal value for each leaf and the overall score of the tree.

XGBoost takes the best parts of AdaBoost and Random Forest to obtain an optimized boosting method:

* Sequential tree growing.
* Minimizing loss function using gradient descent.
* Parallel processing (to increase speed).
* Regularization of parameters.