< Boston House Price Prediction >

작성자 : 김중균

1. 분석 환경

A. Windows 7 64bit(i7-4790, RAM 16G)

B. Python3.6

C. Anaconda

2. 코드 구성

A. Exploratory Data Analysis.ipynb : 데이터 탐색적 분석 과정이 정리된 코드

B. Modeling.ipynb : 예측 모델링을 위한 과정(모델 분석 등)이 정리된 코드

C. Training&Prediction.ipynb : 최종적으로 모델 학습 및 평가가 정리된 코드

   i. Modeling.ipynb 이 각 모델 비교 및 분석이 들어가 있으므로, 여기서 데이터 입력, 변수 선택, 모델 학습, 모델 평가만 추출하여 평가자가 모델을 학습할 수 있도록 따로 구성함.

3. 실행 방법

A. 필요한 library 설치(anaconda 설치 recommended)

   i. Numpy, pandas, scipy, sklearn, matplotlib

   ii. Seaborn (pip install seaborn)

   iii. Xgboost (pip install xgboost)

   iv. Lightgbm (pip install lightgbm)

   v. Mlxtend (pip install mlxtend)

B. Training &Prediction.ipynb 에서 순차적으로 jupyter notebook을 실행하면 됨. (데이터 탐색적 분석과 예측모델링을 위해 여러 모델을 비교 분석하는 부분은 제외되어있음)

4. 탐색적 데이터 분석

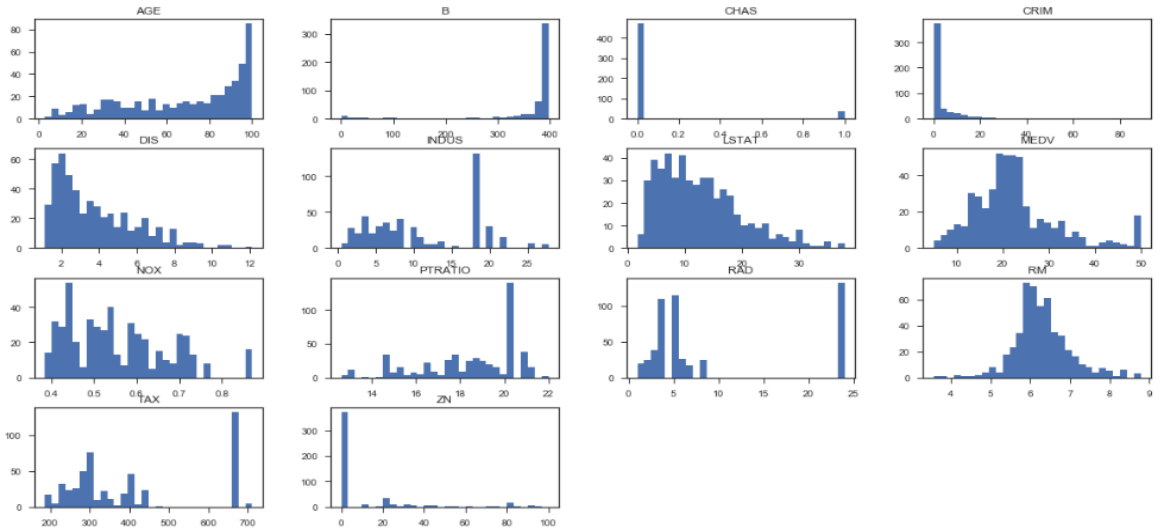
주어진 Boston House Prices 데이터 예측 모델링을 진행 하기에 앞서, 각 변수 별 특징을 탐색해보는 과정을 진행한다. 탐색적 데이터 분석은 다음의 순서로 진행된다.

- 기초통계분석 및 데이터 분포 확인
- 상관관계분석
- 각 변수 별 관계성 파악(Scatter Matrix 활용)
- 종속변수(MEDV)와 상관성 높은 변수 (Scatter Matrix)
- 특정 변수로 그룹화하여 종속 변수와의 관련성 확인
- 변수 별 Skewness 확인

- 기초통계분석 및 데이터 분포 확인

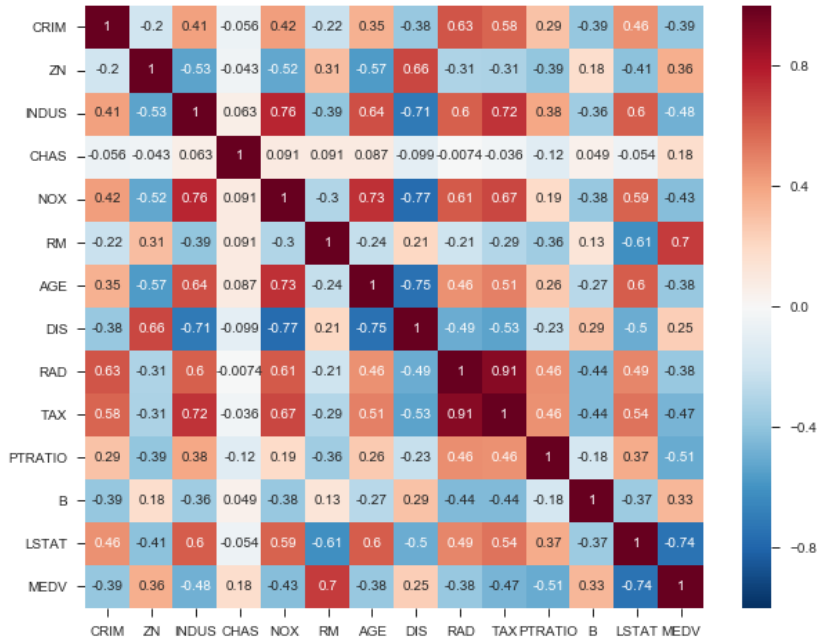
히스토그램을 살펴보면 대체적으로 skewed 되어있음을 확인할 수 있다. 이를 추후에는 skewness를 완화해서 모델링에 활용하는 편이 좋을 것으로 예상된다. 또한 이산형 변수인 CHAS이외에도 ZN, RAD 등은 데이터의 분포가 양분화 시킬 수 있음을 확인 할 수 있다.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	index
count	5.06e+02	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00	506.00
mean	3.61e+00	11.36	11.14	0.07	0.55	6.28	68.57	3.80	9.55	408.24	18.46	356.67	12.65	22.53	252.50
std	8.60e+00	23.32	6.86	0.25	0.12	0.70	28.15	2.11	8.71	168.54	2.16	91.29	7.14	9.20	146.21
min	6.32e-03	0.00	0.46	0.00	0.39	3.56	2.90	1.13	1.00	187.00	12.60	0.32	1.73	5.00	0.00
25%	8.20e-02	0.00	5.19	0.00	0.45	5.89	45.02	2.10	4.00	279.00	17.40	375.38	6.95	17.02	126.25
50%	2.57e-01	0.00	9.69	0.00	0.54	6.21	77.50	3.21	5.00	330.00	19.05	391.44	11.36	21.20	252.50
75%	3.68e+00	12.50	18.10	0.00	0.62	6.62	94.07	5.19	24.00	666.00	20.20	396.23	16.96	25.00	378.75
max	8.90e+01	100.00	27.74	1.00	0.87	8.78	100.00	12.13	24.00	711.00	22.00	396.90	37.97	50.00	505.00

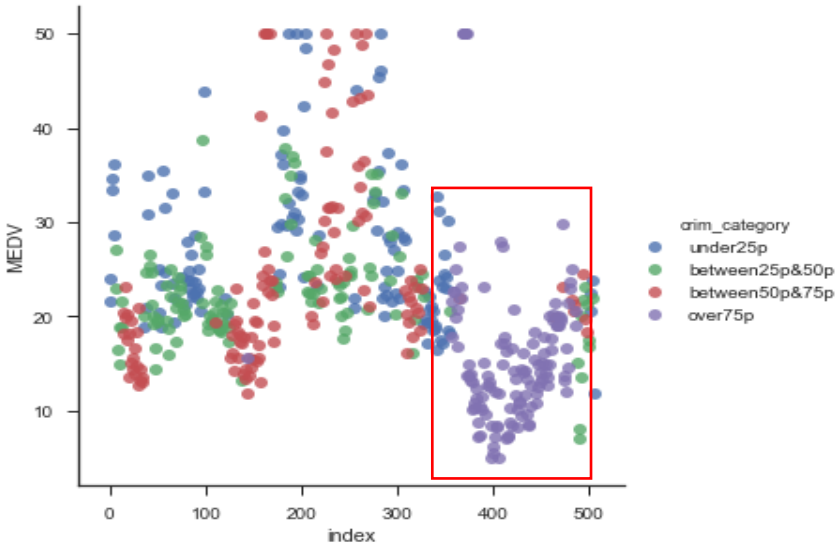


- 상관관계분석

직관적으로 집 가격에 가장 영향을 미칠만한 요소는 CRIM, RM, LSTAT, PTRATIO와 그 다음으로 TAX, DIS, RAD 라 생각하며, 이를 상관관계를 통해 확인한다. 상관관계 결과를 살펴보면, RM과 LSTAT 의 경우 절대값 기준 0.7 이상의 높은 상관성을 보임을 확인 할 수 있다. PTRATIO의 경우도 그 보다는 낮지만 절대값 기준 0.5 이상의 상관성을 보인다. 예상과 달리, CRIM 의 경우는 -0.39로 낮은 상관성을 가지며, 이에 대해 살펴볼 필요가 있다.

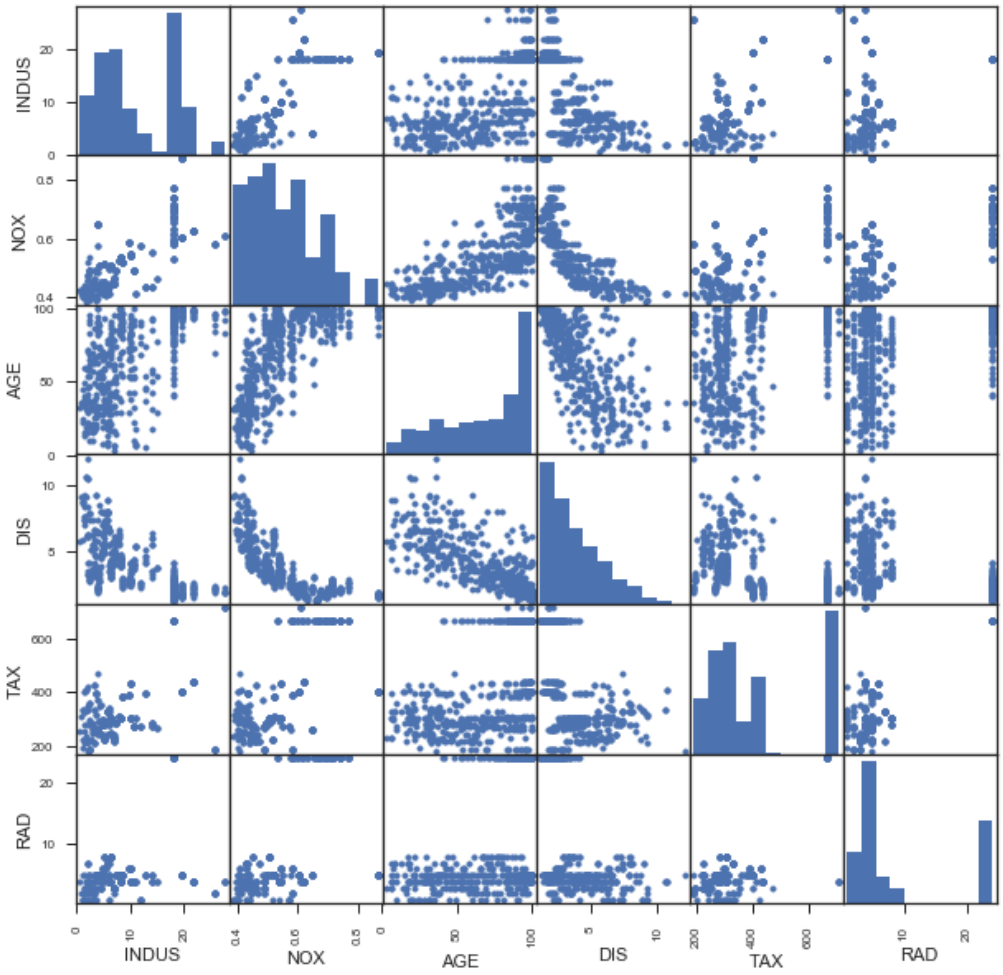


CRIM 데이터의 percentile 별로 총 4개의 그룹으로 나누어 MEDV와의 관계를 살펴본 결과, 75percentile(3.68) 이상은 대부분 30이하의 MEDV의 값을 가짐을 알 수 있다. 상관관계에서는 나타나지 않았지만, 그룹화를 진행하여 살펴보면 CRIM 이 높은 데이터가 집값이 낮게 분포함을 알 수 있다.



- 각 변수 별 관계성 파악 (Scatter Matrix)

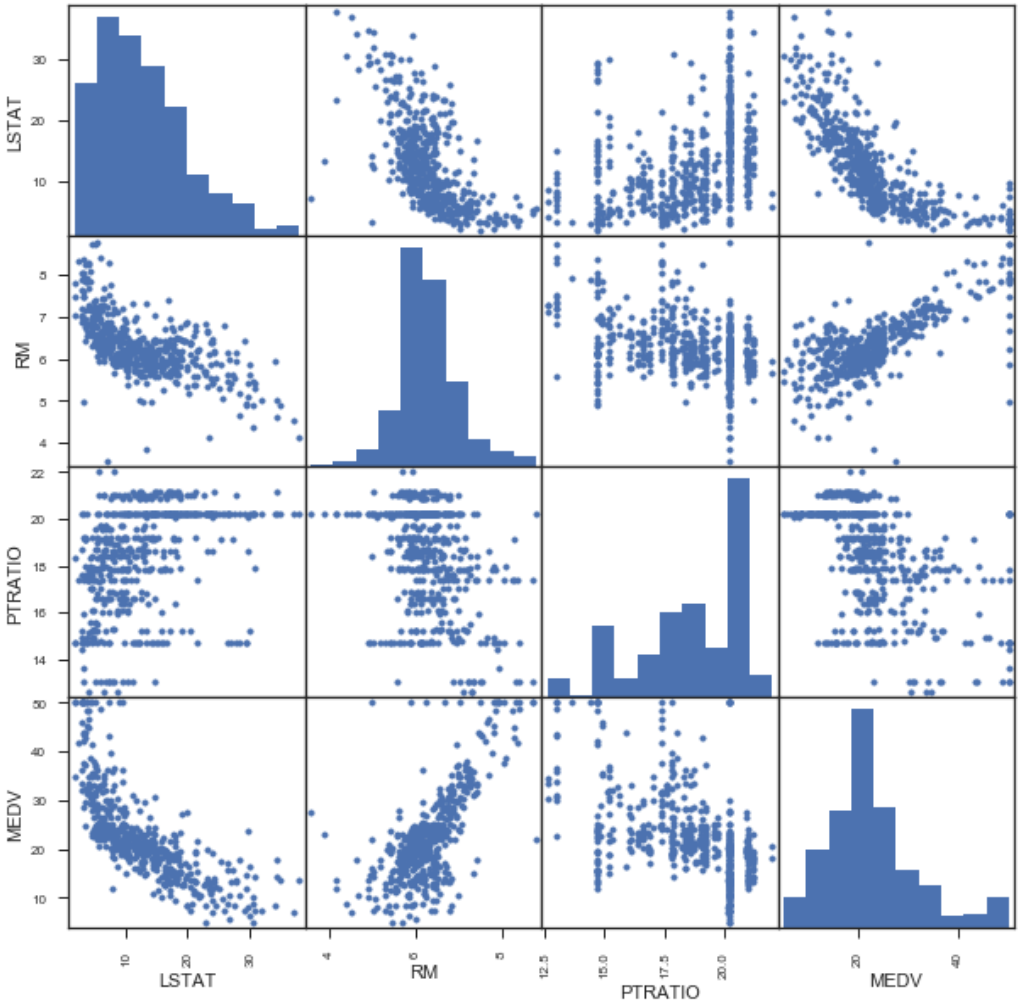
상관관계 결과에서 변수들 간의 상관성이 높은 변수를 추출하여 scatter plot을 통해 데이터 분포를 확인해본다. 상관계수의 절대값이 0.7이상인 변수들을 추출한다. INDUS는 NOX, DIS, TAX와 상관성이 높으며, NOX와 AGE, DIS 그리고 RAD와 TAX 가 상관성이 높다.



TAX와 RAD의 경우 0.91이라는 매우 높은 상관성을 보이는데, 이는 TAX와 RAD가 동시에 높은 값을 가지는 포인트가 모여있기 때문에 높은 상관성을 가짐을 알 수 있으며, 특이하게 TAX와 RAD의 경우 중간의 특정 영역을 기준으로 좌우로 분포가 나뉨을 볼 수 있다. 이러한 데이터 분포에 따라 MEDV가 차이가 발생하는지 그룹화를 통해 확인한다.

- 종속변수(MEDV)와 상관성 높은 변수 (Scatter Matrix)

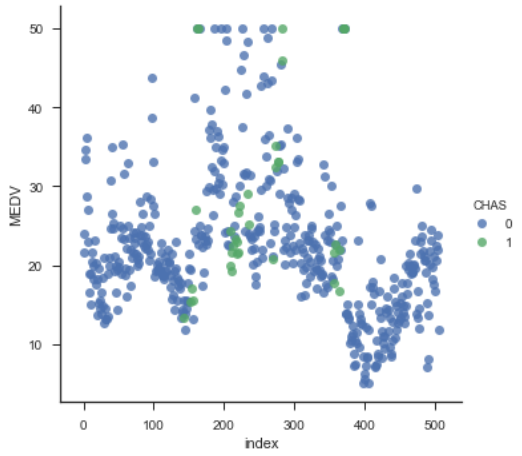
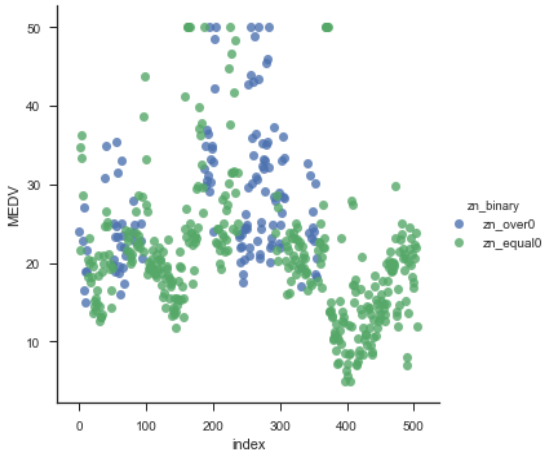
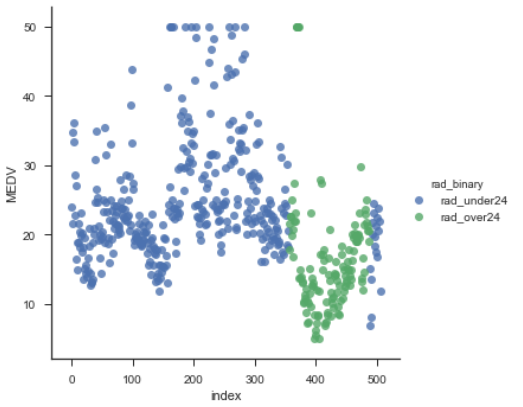
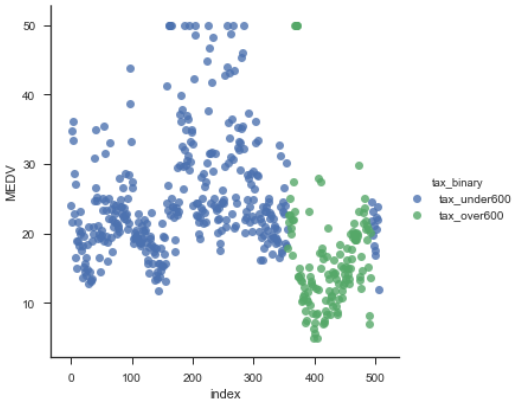
종속변수와 상관성이 상대적으로 높은 변수는 LSTAT, RM, PTRATIO 이며, 이 변수들의 관계를 scatter plot으로 살펴본다.



- 특정 변수로 그룹화하여 종속 변수와의 관련성 확인

기존의 데이터에서 이산형의 역할을 하는 CHAS와, 히스토그램상 데이터 분포가 극명하게 갈리는 ZN, RAD, TAX 변수들로 그룹화하여 종속 변수와의 관계를 확인한다. ZN 변수의 경우, 0의 값을 가지는 데이터가 전체의 50% 이상을 차지하고 있으며, 이를 기준으로 그룹을 나눠 살펴본다. RAD 변수의 경우, max값인 24와 그 아래의 값으로 극명하게 나뉘므로, 이를 기준으로 그룹화하여 변수를 확인한다. TAX 변수의 경우, 600 전후로 극명하게 나뉘므로, 이를 기준으로 그룹화하여 변수를 확인한다. CHAS와 ZN의 경우 값의 분포에 따라 변수들의 뚜렷한 차이점이 보이지 않는다. 하지만 RAD와 TAX의 경우에는 CRIM over 75 percentile과 MEDV scatter plot에서 비슷한 분포를 보이는 것을 확인 할 수 있다. 아래의 표는 세로축은

종속변수인 MEDV 값이며 가로축은 index(data row)를 나타낸다. 아래 결과를 바탕으로 이후 예측모델링에서 이 정보를 변수로써 활용할 수 있다.

CHAS(0 vs. 1)	ZN(over 0 vs. equal 0)
	
RAD (under24 vs. over24)	TAX (under600 vs. over600)
	

- 변수 별 skewness 확인

각 변수들의 skewness를 확인하고 이를 이용하여 Transform 이 필요한 변수를 선택한다. 일반적으로 skewness가 -1보다 작거나 1보다 크면 highly skewed 라고 판단하므로, 이를 확인한다.

CRIM	CHAS	ZN	MEDV	DIS	RAD	LSTAT	NOX	TAX	RM	INDUS	AGE	PTRATIO	B
5.21	3.4	2.22	1.1	1.01	1.0	0.9	0.73	0.67	0.40	0.29	-0.6	-0.8	-2.88

5. 모델링 변수 선정 및 전처리 과정(preprocessing)

모델링 변수 선정 및 전처리 과정은 다음의 순서로 진행된다.

- EDA를 통한 변수 선택
- Transform을 통해 Skewness 완화

- EDA를 통한 변수 선택

앞서 진행한 탐색적 데이터 분석 과정을 통해서 종속변수인 MEDV 를 예측하는데 주요한 변수들을 선택한다.

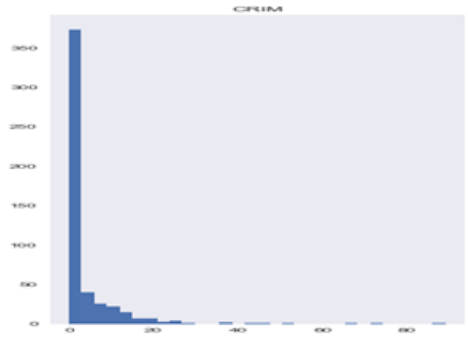
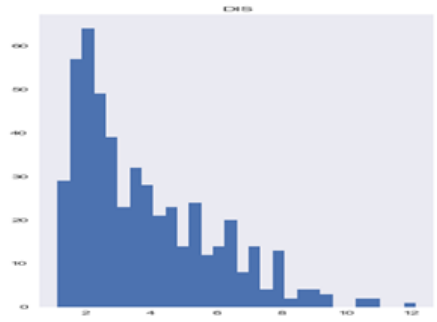
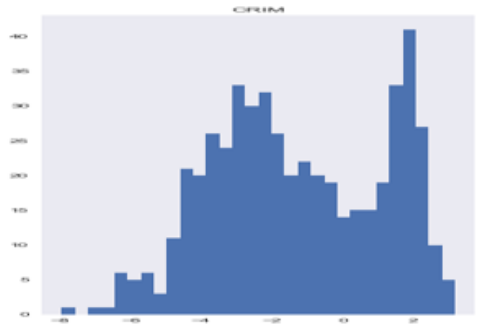
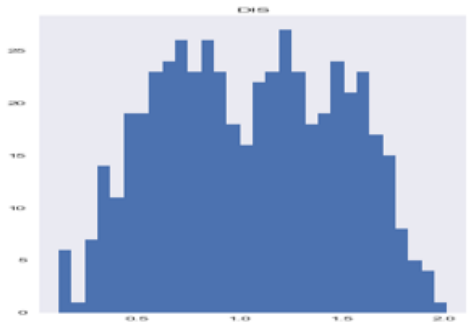
- ✓ MEDV와의 상관관계 분석 결과 연관성 높은 변수 : RM, LSTAT, PTRATIO 선택
- ✓ CRIM의 경우 이 자체는 MEDV와 큰 관련성이 없는 것 같지만 EDA 결과 75 percentile 이상인 데이터에 대해 MEDV 의 경향성이 보이므로 모델에 긍정적 기여가 예상되므로 선택
- ✓ CHAS의 경우 0/1 binary 값에 대해 MEDV의 차이가 눈에 띄지 않으므로 제외
- ✓ ZN의 경우, 그룹화 분석 결과 MEDV와 관련성이 낮으므로 제외
- ✓ TAX와 RAD 의 경우 서로 상관성이 매우 높으므로 이 중 TAX만 선택
- ✓ 나머지 변수 들인 INDUS, NOX, AGE, DIS의 경우 각 변수간의 상관성이 어느 정도 있지만, 매우 높은 편은 아니며, 각 데이터의 산포가 RAD, ZN 보다 크므로 예측 모델링 설명력에 기여할 수 있을 것이라 판단하여 선택
- ✓ B의 경우 MEDV와의 상관성이 제일 낮으며, 다른 변수들과의 관련성도 낮으므로 제외

최종적으로 9개의 변수(RM,LSTAT, PTRATIO, CRIM, TAX, INDUS, NOX, AGE, DIS) 를 예측모델링에 사용될 변수로 선택한다.

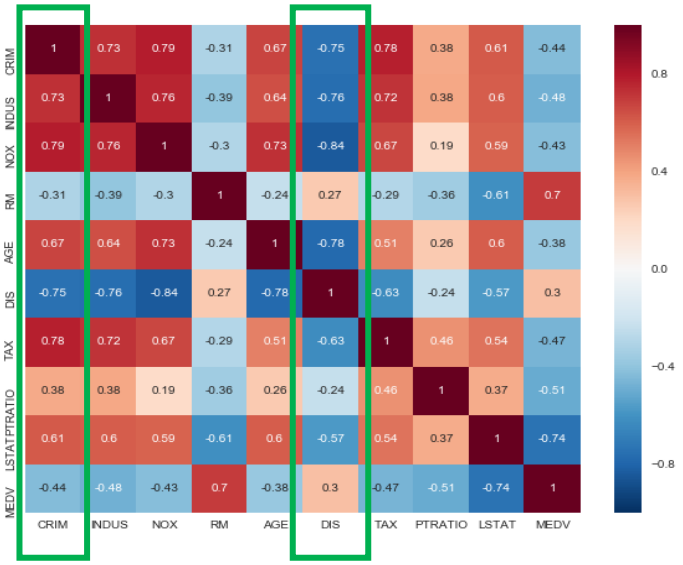
- Transform을 통해 Skewness 완화

Transform을 통해 skewness를 완화한 데이터의 분포를 재확인한다. scipy 에서 지원하는 box-cox transformation을 이용하여 skewness를 완화하는 작업을 진행한다. 각 변수들의 skewness를 확인하고 여기서 절대값이 1보다 큰 변수를 추출한다. 해당 변수들을 boxcox

transformation 을 진행하며, 이때 파라미터로 사용되는 lambda는 optimal 한 box-cox 파라미터를 계산하는 boxcox_normmax 함수를 이용하여 선택한다. CRIM과 DIS 변수가 선택되었으며 이를 변환하면 아래와 같이 분포가 바뀐다.

	CRIM 변수(skewness : 5.2)	DIS 변수(skewness : 1.01)
변환 전		
변환 후		

총 2개의 변수가 선택되어 box-cox 변환을 하였고, 위 아래로 데이터의 분포가 변화했음을 확인 할 수 있다. 변화된 변수를 이용하여 다시 한번 변수 별 상관성을 살펴보면 몇몇 변수들의 상관성이 높아졌음을 볼 수 있으며, 예측모델링 시 다중공선성 문제가 발생할 수 있다.



6. 예측 모델링

예측 모델링 다음의 순서로 진행된다.

- Training/Test 데이터 분할
- 평가 척도 정의
- Cross-validation을 통한 예측모형 성능 비교
- Final Model Training using StackingCVRegressor
- Model Prediction and Evaluation

- Training/Test 데이터 분할

최종적으로 선택된 11개의 변수를 이용하여 예측모델링을 진행한다. 우선적으로 Training 데이터와 Test 데이터를 분할한다. 데이터 분할은 8:2로 진행하며, 분할된 Test 데이터는 최종적으로 생성된 모델의 성능을 검증하는데 활용된다.

- 평가 척도 정의

해당 문제의 평가 척도로는 RMSE(Root Mean Square Error) 를 이용한다. 이는 실제 Kaggle에서 제시하는 평가척도이다.

- Cross-validation을 통한 예측모형 성능 비교

우선 크게 Simple regression model과 ensemble model을 나눠서 모델 별 성능을 비교 분석하고, 최종적으로 활용할 모델을 선택한다. 모델 별 성능을 비교하기에 앞서 각 모델별로 search 과정을 거쳐 최적의 hyper parameter 를 선택한 후, 각 모델이 최적화 된 상태에서 비교 분석을 진행한다.

모델링을 진행할 때, Training 데이터에서 k-fold cross-validation 을 이용하여 모델의 평균적인 성능을 확인하며, hyper parameter tuning에도 활용된다. cross-validation에 사용될 fold의 수는 10개로 하며, 총 10번의 서로 다른 데이터에 대한 모델 학습 및 도출된 학습 결과의 평균을 통해 예측모형들의 성능을 비교한다. 첫 번째 단계로, 예측 모델을 적용하기 이전에 데이터 표준화를 진행한다. 데이터마다 스케일이 다르기 때문에, 이를 보정하여 예측모델의 성능을 높이고자 한다. 이를 수행하기 위해 sklearn.preprocessing 에

서 StandardScaler를 이용한다. 데이터표준화와 모델의 구성을 합치기 위해 파이프라인을 이용하여 구성한다. 모델 성능 평가 지표로는 RMSE를 사용한다.

Cross-validation을 통한 hyper parameter tuning 에 본 분석에서 사용하고자 하는 RMSE를 적용하기 위해서는 아래와 같은 코드가 필요하다. Make_scorer란 함수를 통해 RMSE를 적용해주어야 한다. 하지만 search 내부적으로는 score를 maximize하는 방향으로 찾기 때문에 RMSE가 음수가 발생하게 되므로, score를 출력할 때는 (-)를 곱해주어야 한다.

```
# Gridsearch를 위해 rmse 를 평가척도로 따로 설정함. 기본적으로 gridsearchcv 는 score를 maximize하는 방향으로 학습되므로,
# RMSE를 활용하기 위해서는 greater_is_better=False 로 설정한다.
# RMSE로 진행하더라도 내부적으로 score를 음수로 반환하므로 여기에 -를 곱해주어야한다.
scoring_fn = make_scorer(cal_rmse, greater_is_better=False)
```

본 분석에서는 모델의 hyper parameter tuning을 위해 Randomized Search와 Grid Search를 동시에 고려하는 search를 제시한다. 일반적으로 Grid search 혹은 Randomized search 중 하나를 사용하지만, 어떤 search가 더 좋은 성능을 나타내는지 모르기 때문에 두 가지 search를 모두 활용할 수 있도록 구성한다. 'fit_optimal_model' 라는 custom 함수를 통해 각 모델에서 성능이 높은 hyper parameter를 찾고, 최적의 hyper parameter로 조정된 모델로 pipeline을 구성한 결과를 얻는다. 함수에 대한 설명은 아래와 같다.

```
def fit_optimal_model(model,params, x, y, n_folds, seed):
    # 설명 : Randomized search 와 Grid search를 동시에 활용해서 모델의 가장 좋은 parameter를 찾는다.

    # -----Input 설명--------
    # model은 name, pipeline 으로 받는다.
    # param은 model parameter set
    # x : X training set
    # y : Y training set
    # n_folds = k-fold의 n
    # seed : random seed
    # ----------------------

    # -----return 설명---------
    # best_model : Grid & Randomized search 결과 성능이 가장 좋은 모델을 반환(pipeline으로)
    # best_param : Grid & Randomized search 결과 성능이 가장 좋은 모델의 parameter set을 반환
    # best_search : Grid & Randomized search 결과 성능이 가장 좋은 search 의 결과물을 반환
```

우선적으로 먼저 사용할 예측모형들은 Lasso, Ridge, KNN, CART, SVR, LinearRegression 을 이용한다. 앞서 데이터 전처리를 통해 다중공선성 문제의 발생 가능성을 엿봤기 때문에, 이를 제어할 수 있는 예측모형인 Lasso, Ridge 의 결과를 통해 다중공선성 문제가 결정적으로 작용하는지 확인한다. 또한 전통적으로 많이 사용되는 KNN, CART, SVR, LinearRegression 도 비교한다. 먼저 각 모델 별 hyper parameter를 정리한다.

- ✓ LinearRegression : 유일한 parameter인 fit_intercept(True or False)
- ✓ LASSO : L2 regularizer의 가중치인 alpha(아주 작은 1e-4 ~ 1보다 큰 10까지 검증)
- ✓ RIDGE : L1 regularizer의 가중치인 alpha(아주 작은 1e-4 ~ 1보다 큰 10까지 검증)
- ✓ KNN : neighbor의 수(3~10)
- ✓ CART : 트리의 max_depth, split을 하기 위한 최소한의 sample수인

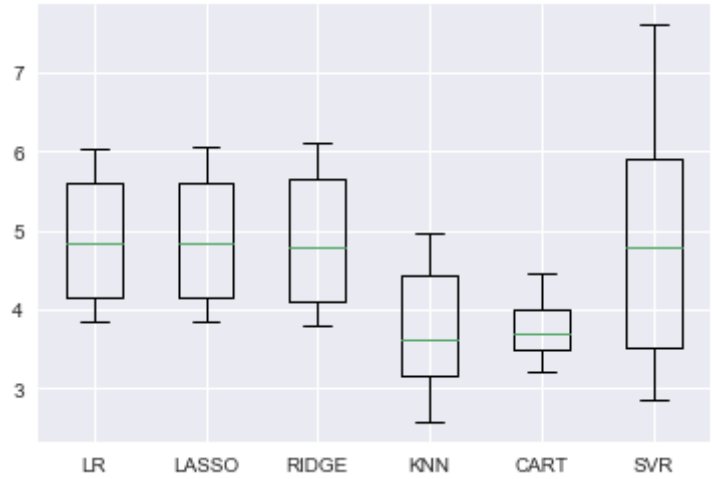
min_samples_split 을 고려

- max_depth의 경우 데이터 수가 적으므로 매우크지않는 선에서 설정 (6~10)
- min_samples_split 또한 traning data가 cross-validation 시 400개 정도밖에 되지 않으므로, 작은 수치로 설정 (2~5)

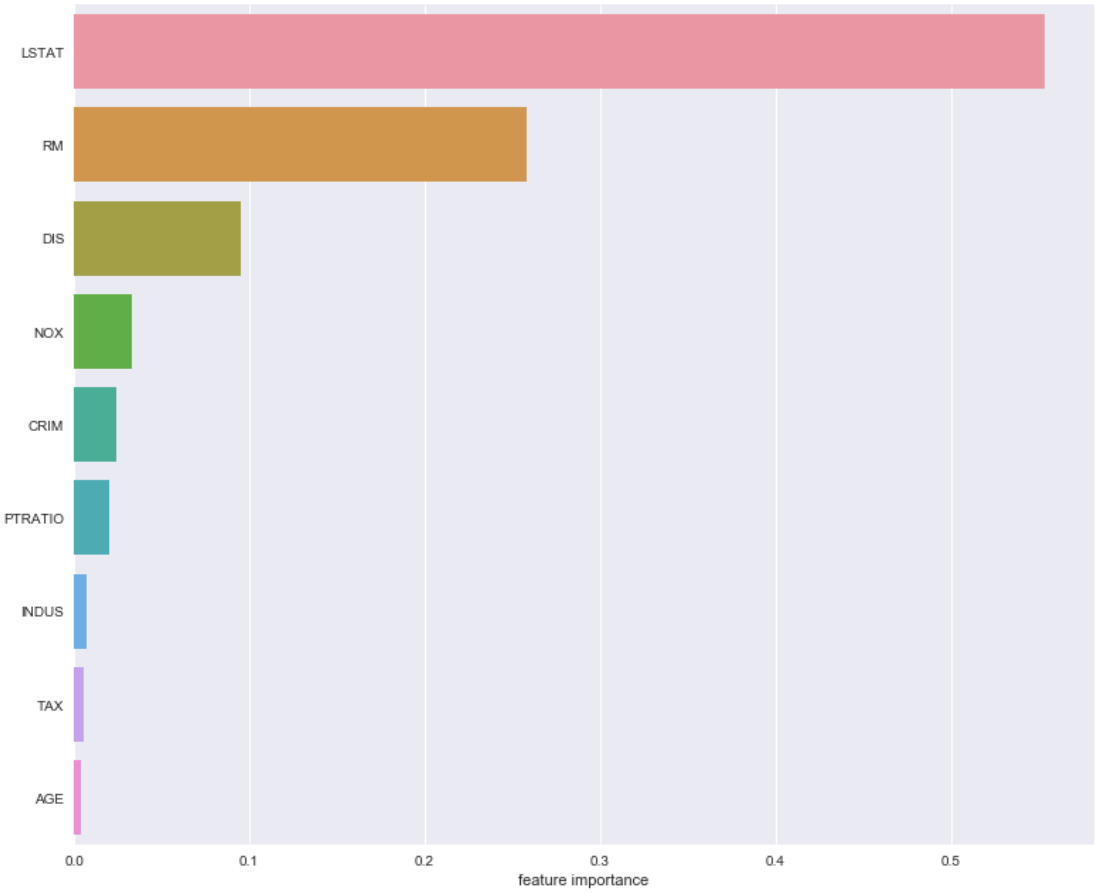
✓ SVR : kernel 종류 (RBF, poly, sigmoid)

6개의 모델에서 현재 training data 에서 최적의 hyper parameter를 찾고, 찾은 최적 hyper parameter cross-validation 한 결과를 출력한다. 이를 box-plot을 통해 평균치와 분산을 확인한다. 그 결과는 아래와 같다. 10 fold cross-validation LinearRegression과 LASSO, RIDGE 이 거의 비슷한 결과를 나타냄을 알 수 있다. LASSO와 RIDGE 는 각각 L1, L2 Reguralizer를 활용하여 overfitting을 피하고 다중공선성 문제를 완화시킨다고 알려져 있다. 이전의 Box-Cox 변환을 통해 변화된 변수들의 상관성을 확인한 결과, 다중공선성 문제의 여지가 있다고 분석했는데 위의 모델 성능을 봤을 땐 크게 문제가 없는 것으로 보인다. 또한 간단한 6개의 모델 결과에서는 KNN과 CART가 가장 좋은 성능을 보임을 알 수 있다. KNN의 경우 n이 3~4에서 좋은 결과를 보이며, CART의 경우에서도 max_depth가 비교적 적은 수치(6 전후)에서 높은 성능을 보였다. 데이터의 수가 적고 단순하기 때문에 굳이 큰 사이즈의 Tree가 필요가 없을 것으로 예상된다.

	Linear Regression	LASSO	RIDGE	KNN	CART	SVR
RMSE 평균 (표준편차)	4.890677 (0.776614)	4.890638 (0.777008)	4.882848 (0.817652)	3.713631 (0.817542)	3.688043 (0.464919)	4.839358 (1.457081)
	fit_intercept	Alpha	Alpha	n_neighbors	max_depth/ min_samples_split	Kernel
Hyper parameter	True	0.001	8.2786	4	6/5	rbf



CART 의 경우는 아래와 같이 변수 별 중요도를 추출할 수 있다. 이를 통해서 MEDV와 가장 상관성이 높은 LSTAT와 RM이 가장 큰 중요도를 차지하고, DIS, NOX가 다음이었다. CRIM의 경우 EDA에서 분석한 것처럼, 특정 영역에서 중요한 정보를 갖고 있으므로, 예측모델링에 기여했다고 분석 할 수 있다.



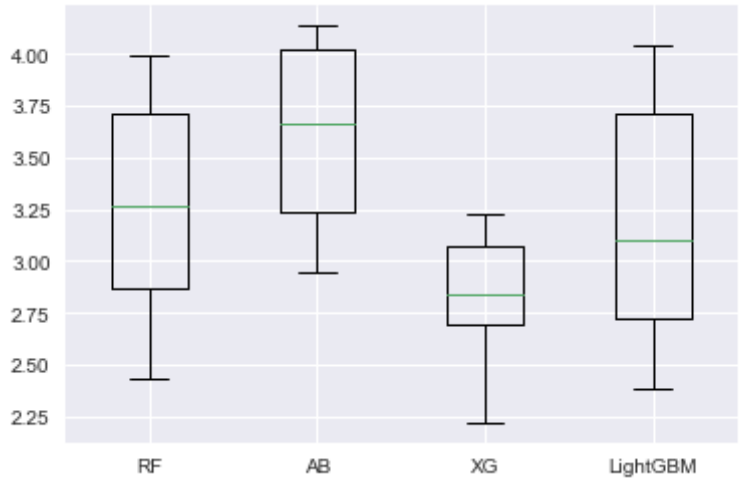
간단한 Regression 모델에 이어서 앙상블 모델을 위와 같이 동일하게 적용하여 모델의 성능들을 비교해본다. 여기서 사용할 앙상블 모델은 RandomForest, AdaBoost, XGboost, LightGBM 을 사용한다. 마찬가지로 각 모델들의 파라미터를 정리한다.

- ✓ 공통 파라미터 : n_estimators 는 공통적으로 100~300으로 제한한다. step은 50으로 설정하고, randomized search에서는 100~300사이의 랜덤값을 이용한다. 또한 Randomforest를 제외하고 learning rate를 사용하는데 이는 1e-4 ~ 1e-1로 설정한다. 일반적으로 좋은 학습 성능을 이루기 위해서는 작은 learning rate를 추천하기 때문에, 작은 수치인 0.1보다 작은 수치로 실험

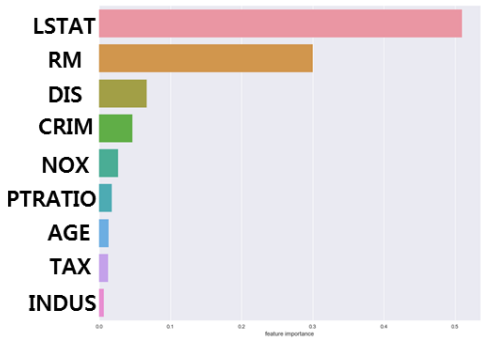
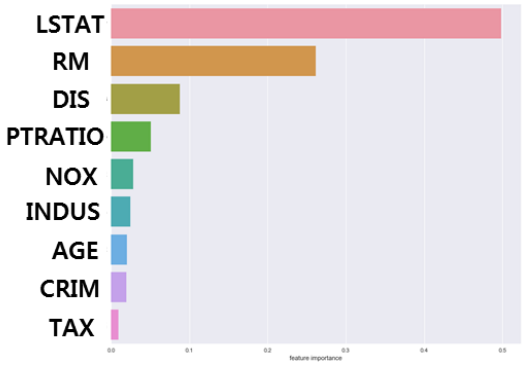
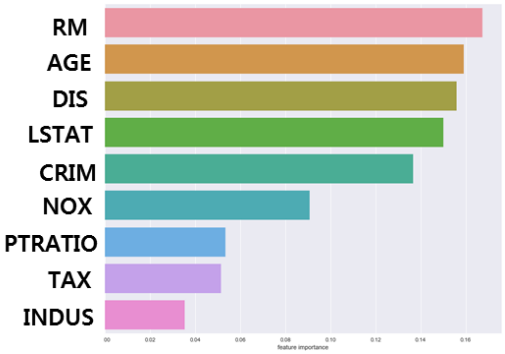
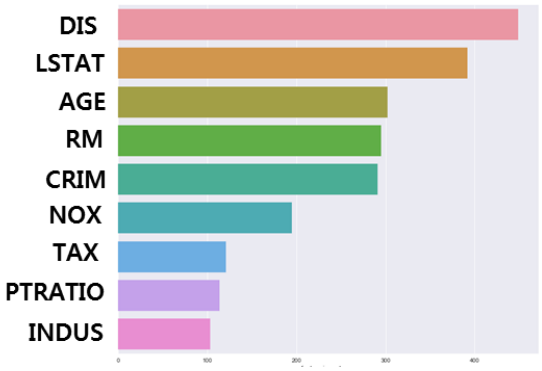
- ✓ RandomForest : 앞서 DecisionTree에서 설정했던 parameter는 동일하게 설정하고, 추가로 bootstrap 옵션을 고려 (True or False)
- ✓ Adaboost : loss를 고려(linear, square, exponential)
- ✓ XGboost : max_depth는 위의 DecisionTree와 동일하게 구성하며, 추가로 min_child_weight, colsample_bytree, gamma를 고려
  - ■ min_child_weight : chlid에서 필요한 모든 관측치에 대한 가중치의 최소 합에 대한 제약으로, 큰 값은 over-fitting을 방지 (1~5)
  - ■ colsample_bytree : 개별 트리를 구성할 때 칼럼들이 랜덤하게 샘플되는 비율을 말하며, 일부 샘플링을 위해 0.8~1.0으로 설정
  - ■ subsample : 개별 트리를 구성할 때 traning data가 랜덤으로 샘플되는 비율을 말하며, 0.8 ~ 1.0 으로 설정
- ✓ LightGBM : 공통 parameter 이외에 num_leaves, boosting 옵션을 고려(num_leaves < 2^(max_depth) 이므로 이 제약을 만족하는 num_leaves를 찾기 위해 max_depth를 7로 고정)
  - ■ num_leaves : 트리에 존재하는 총 leaves의 수로, 성능에 아주 중요한 역할을 함 (40~80)
  - ■ boosting : boosting 방법 중 선택하는 것으로, dart는 Dropouts meet Multiple Additive Regression Trees 를 의미함 (gbdt(default) or dart)
  - ■ LightGBM hyper parameter tuning의 경우 공식 document 에 recommended 된 옵션을 위주로 실험

앙상블 모델의 경우 위에서 실험한 간단한 regression 모델에 비해 전체적으로 더 높은 성능을 나타낸다. CART가 가장 높은 성능을 나타냈지만 Adaboost가 더 높은 성능을 보인다. 앙상블 모델 중 가장 좋은 성능을 보이는 모델은 XGBoost로 학습시간에서는 상당한 시간이 걸리지만 좋은 성능을 보임을 알 수 있다. 또한 이전의 CART의 결과와 비슷하게 max_depth는 모두 6정도로 깊은 트리는 필요하지 않음을 알 수 있다.

	RandomForest	Adaboost	XGboost	LightGBM
RMSE평균 (표준편차)	3.265418 (0.504214)	3.607081 (0.419985)	2.879215 (0.402816)	3.200663 (0.580637)
Hyper parameter	bootstrap: True, max_depth: 6, min_samples_split: 2, n_estimators: 100	learning_rate : 0.1, loss : 'linear', n_estimators : 100	colsample_bytree : 0.8, learning_rate : 0.1, max_depth : 4, min_child_weight : 3, n_estimators : 150, subsample : 0.9	Boosting : 'gbdt', learning_rate : 0.1, n_estimators : 200, num_leaves : 40

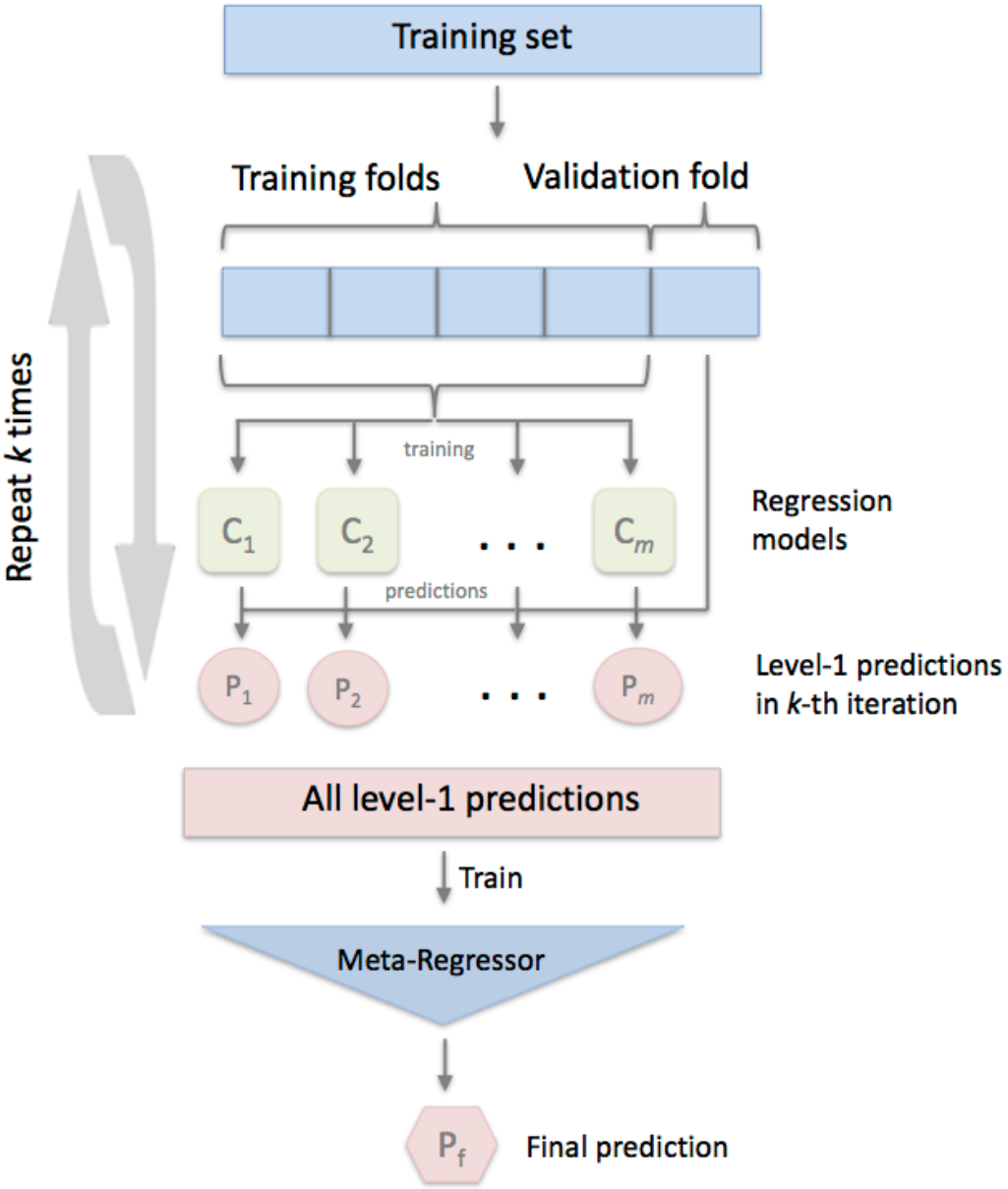


각 앙상블 모델 별로 feature importance를 살펴보면, 종속변수와 상관성이 높은 LSTAT와 RM은 항상 상위에 위치해있음을 알 수 있다. 상대적으로 성능이 좋은 XGboost와 LightGBM의 경우는 importance 기준 상위 5개 변수가 크게 영향력을 미침을 알 수 있다. XGboost와 LightGBM 이 다른 두 모델과 명확이 차이 나는 변수는 AGE, DIS, CRIM으로 기존의 EDA 분석에서 느꼈던 중요도보다 더 중요한 변수임을 확인할 수 있다.

RandomForest	Adaboost
	
XGboost	**LightGBM**
	

- Final Model Training using StackingCVRegressor

최종적으로 mlxtend 라이브러리의 StackingCVRegressor 라는 앙상블 방식을 이용하여 최종 모델을 학습한다. StackingCVRegressor 이란 meta-regressor를 활용하는 앙상블 방법으로, 총 2가지의 레벨로 나뉜다. 첫번째 레벨에서는 일반적인 regressor(base regressor)를 이용하여 Training Data를 K-fold로 분할하여 K-1 로 학습을 하고 남은 하나의 셋에 대해 예측한다. 예측한 결과물들을 두 번째 레벨의 meta-regressor의 meta-feature로 활용하고, 선택적으로 original data를 추가로 활용하여 최종적으로 예측모형을 학습하게 된다. 아래는 StackingCVRegressor 학습 방식을 표현한 그림이다.



(출처 : http://rasbt.github.io/mlxtend/user_guide/regressor/StackingCVRegressor/)

다음은 StackingCVRegressor를 이용하여 새롭게 만든 stack_train 함수의 학습 절차를 나타낸 pseudo-code 이다..

```
# Choose base regressor

For all regressor r:

    Append best tuning model using 'fit_optimal_model' custom method

    to best_model_list

sort best_model_list by performance

choose n regressors as base regressor


# StackingCVRegressor Training Procedure

For each base regressor b:

    For iter in cross-validation:

        fit b to training k-1 folds

        predict k-th fold(hold-out fold) using b; aka meta-features

use meta-features(optionally with original data) to fit meta-
regressor

fit each base regressor to the original data
```

새롭게 만든 stack_train은 위에서 정의한 fit_optimal_model 을 이용하여 성능이 높은 n개의 모델 선택하여 이를 base regressor로 활용한다. meta-regressor로 RandomForest를 활용하여 Stacked Regressor를 학습한다. fit_optimal_model을 통해 최적의 base regressor 모델을 찾으면 해당 base regressor는 고정하고, meta-regressor를 grid search로 hyper parameter tuning을 진행한다. Base model이 어느정도 tuning이 되어 있으므로, meta-regressor에서의 hyper parameter tuning은 최소한으로 진행한다. 여기서는 n_estimators와 max_depth만을 이용한다.

```
def stack_train(X, Y, models, params, n_folds, seed, n_choice_model = 3):

    # 설명 : 위에서 정의한 fit_optimal_model 을 이용하여 성능이 높은 n_choice_model 를 선택하여 이를 base regressor로 활용하고,
    #        meta-regressor로 RandomForest를 활용하여 Stacked Regressor 를 학습한다. fit_optimal_model을 통해 최적의 base regressor
    #       모델을 찾으면 해당 base regressor는 고정하고, meta-regressor를 grid search로 hyper parameter tuning을 진행한다.
    #
    # -----Input 설명--------
    # x : X training set
    # y : Y training set
    # model은 name, pipeline 으로 받는다.
    # param은 model parameter set
    # n_folds = k-fold의 n
    # seed : random seed
    # n_choice_model : Base regressor로 활용할 모델의 수
    # ----------------------

    # -----return 설명--------
    # best_model : Grid search 결과 성능이 가장 좋은 StackingCVRegressor 을 반환
    # selected_model : fit_optimal_model 결과 성능이 높아서 base regressor로 선택된 모델의 이름을 반환
    # best_base_model : 선택된 base regressor 중 가장 성능이 좋은 regressor 반환
```

본 분석에서 StackingCVRegressor를 활용한 이유는 다음과 같다.

- ✓ 일반 average ensemble 방식은 각 모델의 예측치를 linear combination으로 결합하여 최종 결과를 내는데, 이에 비해 meta-regressor를 이용함으로써 더 좋은 성능을 낼 것이란 기대
- ✓ Meta-regressor 가 각 base model의 예측치를 input으로 받으면서 기존 base model 이 설명하지 못하는 잔차를 한번 더 학습한다는 개념
- ✓ 여러 개의 model을 base regressor로 이용하면서 하나의 모델로 fine-tuning 한 결과보다 Test 데이터에 대해 robust한 결과를 도출할 것이라는 기대

- Model Prediction and Evaluation

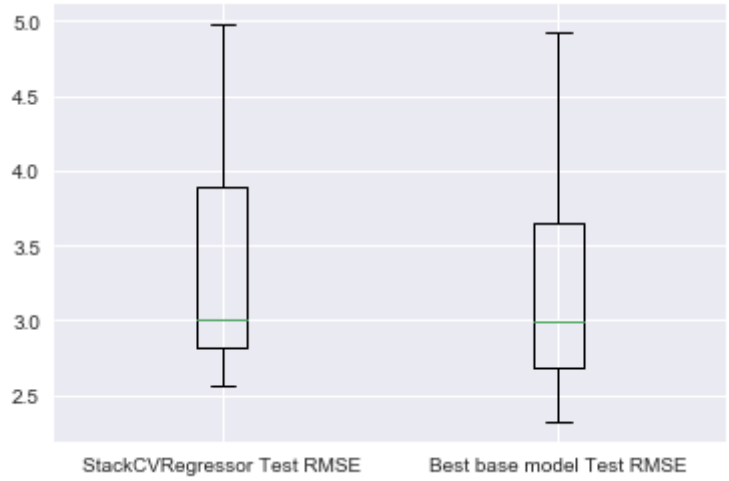
최종적으로 위에서 실험한 간단한 regression 모델과 ensemble 모델을 합쳐서 StackingCVRegressor를 학습한다. 총 10의 candidate model 이 주어지며 본 분석에서는 3개의 base model만을 선택한다. 학습된 StackingCVRegressor를 이용하여 이전에 분할해 놓은 test 데이터로 결과를 얻는다. 최종 결과는 RMSE로 계산된다.

학습 결과 다음과 같다. 사용된 base model은 hyper parameter tuning 결과 XGboost, LightGBM, RandomForest가 선택되었고, 이를 이용하여 최종적으로 학습된 StackingCVRegressor은 RMSE가 2.97로 XGboost 보다 낮은 성능이 도출되었다. 하지만 Test 데이터에 대해서는 XGboost보다 높은 성능을 보임을 확인 할 수 있다.

	Model Name	RMSE
Base Model	XGboost	2.896
	LightGBM	3.195
	RandomForest	3.195
StackingCVRegressor	StackingCVRegressor	2.970
	With RandomForest meta-regressor hyper parameter set max_depth : 5, n_estimators : 150	
StackingCVRegressor Test RMSE	2.7904	
Best Base Model(XGboost) Test RMSE	2.8928	

또한 학습하는 프로세스를 다 함수로 구성해놓았기 때문에, Training/Test 데이터를 여러 번 split하면서 모델의 평균적인 성능을 검증할 수 있다. 총 5번의 random seed를 통해서 Training/Test를 구분하고 학습과정을 거쳐 최종 결과를 출력한다. 아래는 학습한 StackCVRegressor 와 StackCVRegressor를 학습하기 위해 선택된 Base model 중 가장 성능이 좋았던(모든 결과에서 XGBoost가 선택됨) 모델의 Test 데이터에 대한 RMSE 결과이다. 5

번의 서로 다른 학습 결과, 한번을 제외하고 XGboost의 성능이 높았다.



Random seed	0	1	2	3	4
StackCVRegressor	4.975859	3.008291	**2.569045**	2.817315	3.894496
Best Base model (XGboost)	**4.918751**	**2.687349**	2.998607	**2.322416**	**3.648829**