

# House Price Predict

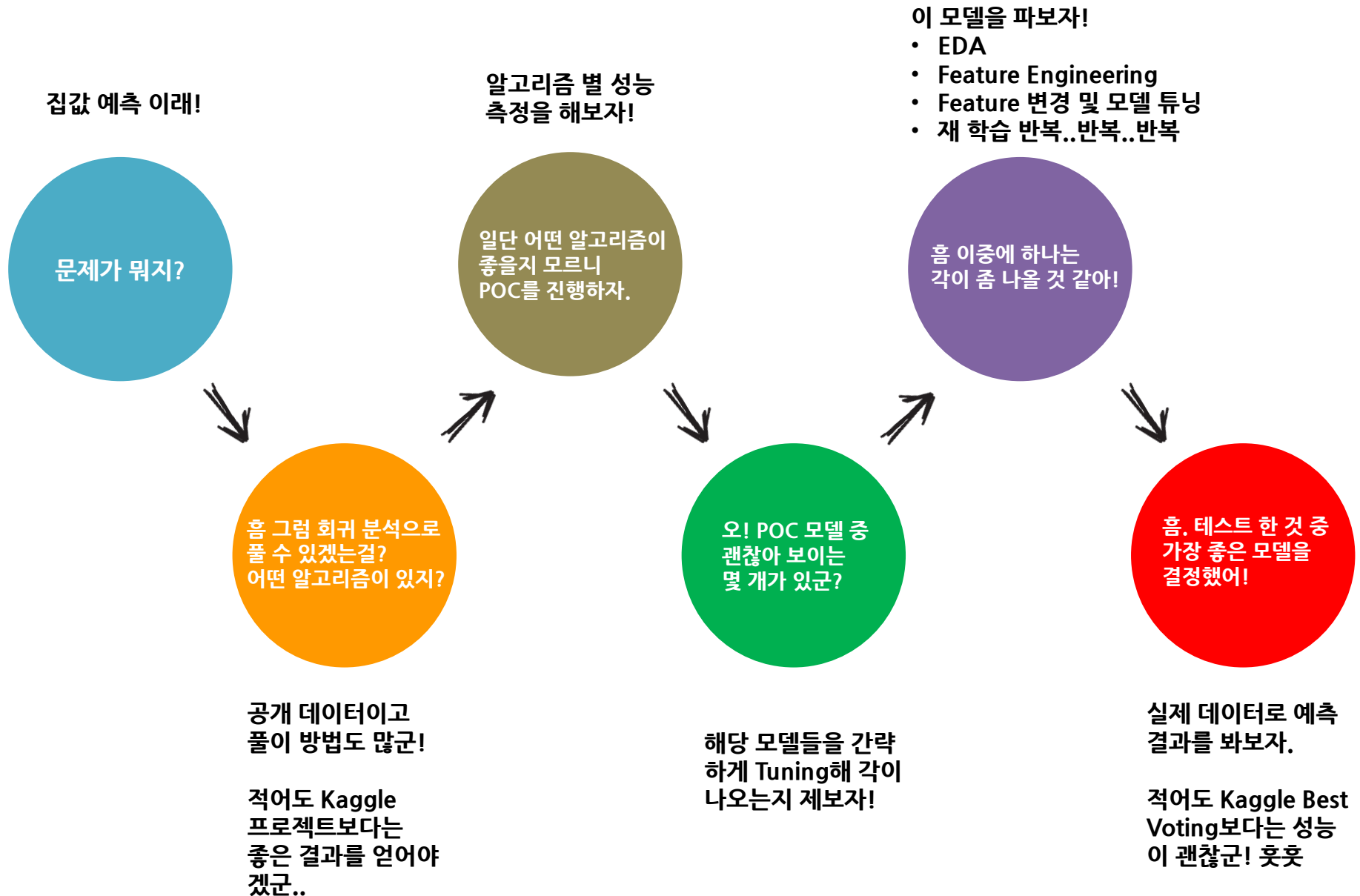
---

이웅규

해당 문서는 제출한  
Jupyter notebook을 기반으로 작성하였으며  
문제를 해결하는 것보다  
문제를 **어떻게** 해결하고자 했는지에 대한  
**생각의 흐름**을 작성한 문서입니다.

<https://github.com/teriusbin/boston-house-price-predict>

# 생각의 흐름



# 목차

## 1 데이터 로드 및 파싱

## 2 기본 모델 선정하기

## 3 POC 모델 개선하기

- 2.1 EDA
  - 2.1.1 기초 데이터 확인
  - 2.1.2 데이터 분포 확인
- 2.2 Feature Scaling (origin & std & norm)
- 2.3 학습 데이터 셋 / 테스트 데이터 셋 분리
- 2.4. POC 모델 만들기

- 3.1 EDA
  - 3.1.1 Feature간 상관 관계
- 3.2 Feature Engineering
- 3.3 학습 데이터 셋 / 테스트 데이터 셋 분리
- 3.4 Input별 성능 비교
- 3.5 모델 튜닝
  - 3.5.1 Gradient Boost 모델 튜닝
  - 3.5.2 DNN 모델 튜닝

- 4.1 Hidden Layer 증가에 따른 모델 튜닝
  - 4.1.1 Hidden Layer 5개
  - 4.1.2 Drop Out 추가
  - 4.1.3 Regularization 추가
  - 4.1.4 학습 시간 조정하기
  - 4.1.5 가중치 초기화
  - 4.1.6 Batch Normalization 추가
  - 4.1.7 Learning Rate 변경
  - 4.1.8 Mini Batch Size 조정
  - 4.1.9 학습 시간 조정하기
- 4.2 Hidden Layer 감소에 따른 모델 튜닝
  - 4.2.1 가중치 초기화
  - 4.2.2 Learning Rate 변경
  - 4.2.3 Drop Out 비율 변경
  - 4.2.4 Early Stopping
- 4.3 Feature 추가 및 변경에 따른 모델 튜닝
  - 4.3.1 EDA
  - 4.3.2 Feature Engineering
    - 4.3.3.1 범주형 Encoding Feature
    - 4.3.3.2 원본 Feature
  - 4.3.3 학습
- 4.4 Attention Mechanism
  - 4.4.1 Feature Engineering
  - 4.4.2 Attention Layer 학습 (Noise Feature 추가)
  - 4.4.3 활성화 층 확인
  - 4.4.4 Attention Layer 학습
  - 4.4.5 활성화 층 확인
  - 4.4.6 Feature Selection
  - 4.4.7 모델 학습

- 5.1 최종 모델
- 5.2 평가 및 측정
- 5.3 아쉬운 점

## 4 최종 모델 Deep Dive

## 5 결과

해당 문서는 제출한  
Jupyter notebook을 기반으로 작성하였으며  
문제를 해결하는 것보다  
문제를 **어떻게** 해결하고자 했는지에 대한  
**생각의 흐름**을 작성한 문서입니다.

## 0. 환경 및 필요 Library

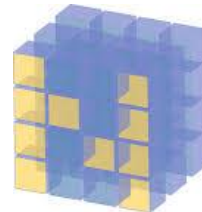
seaborn

matplotlib  Version 3.0.3

 Keras

 scikit  
*learn*

Pandas 

 NumPy

 python<sup>TM</sup>

 jupyter

# 0. 필요 Library 및 실행 방법

## 0. 필요 library Import

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas import DataFrame, Series
import keras
from keras import layers
from keras import models
from keras.models import Model
from keras.layers import Dropout
from keras import regularizers
from keras.models import model_from_json
from keras.optimizers import Adam
from keras.layers.normalization import BatchNormalization
from keras.layers import Activation
from keras.wrappers.scikit_learn import KerasRegressor
from keras import backend as K
from sklearn.model_selection import cross_val_score
from sklearn.cross_validation import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn import ensemble
from sklearn import datasets
```

- <https://github.com/teriusbin/boston-house-price-predict>
- 전체 프로젝트 실행 방법
  - 상위 링크에서 boston\_house\_predict.ipynb 파일을 다운
  - Jupyter 환경에서 해당 notebook을 차례로 실행
- 최종 모델에 대한 예측을 원할 경우
  - 상위 링크에서 model.h5, model.json 파일을 다운
  - Jupyter notebook 마지막 5. 결론 하위 부분에 있는 내용을 차례로 실행

# 1. 데이터 로드 및 파싱

- 명시된 Column을 정의
- 첨부된 데이터 저장소 링크를 이용해 Pandas Dataframe으로 로드
- 데이터 shape, 데이터의 구성을 확인
- 명시된 Column의 의미 파악

	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

- 컬럼 정보
  - CRIM - Town별 범죄 비율
  - ZN - 25,000피트를 넘는 거주지역 비율
  - INDUS - 비 상업지역의 비율
  - CHAS - 찰스강 강둑 주변이면 1, 아니면 0
  - NOX - 일산화 질수 수치
  - RM - 방 수
  - AGE - 1940년 이전에 건축된 주택 비율
  - DIS - 보스턴 직월센터의 접근성
  - RAD - 고속도로 접근성
  - TAX - 재산세
  - PTRATION - 학생 / 교사 비율
  - B - 흑인 비율
  - LSTAT - 하층민 비율
  - MEDV - 주택 가격



## 2. 기반 모델 선정하기

### 목적

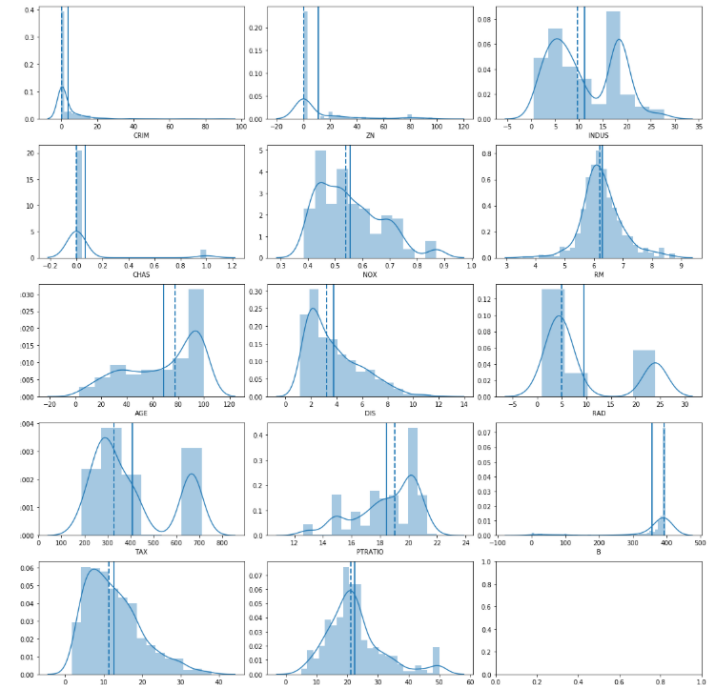
---

회귀 문제를 풀 수 있는 여러 모델 중  
어떤 모델을 선정할지에 대한 후보군 추출 **POC**

## 2. 기반 모델 선정하기

### 2.1 EDA

- 기초적인 데이터 확인
- **WHY? 데이터의 전처리 유무를 결정하기 위함**
  - Null 체크
  - 데이터 타입 확인
  - 데이터 중복 여부 확인
- 데이터의 분포 확인
- **WHY? 데이터 분포를 통해 Feature별 특성 확인**
  - 데이터의 기초 통계량 확인
  - 데이터의 분포 가시화
- EDA 결과
  - MEDV와 RM Feature의 경우 정규 분포 형태로 데이터가 분포함
  - 정규분포로 변경 가능한 데이터들은 추 후 분포를 변형할 필요가 있다고 판단됨
  - CRIM, ZN, B의 경우 지나치게 Skew 되어 있기 때문에 별도의 의미 파악이 필요
  - INDUS, NOX, RAD, TAX, PTRATION도 특정 값에 치우쳐 있는 형태를 보임. 추가 파악 필요



## 2. 기반 모델 선정하기

### 2.2 Feature Scaling (Origin & Standardization & Normalization)

- WHY?

- 데이터의 분포만으로는 학습에 필요한 데이터의 경향성을 파악하기 힘들다고 판단
- 3 종류의 Input Feature를 만들고 POC 모델들의 학습 결과를 통해 적절한 Input 형태를 결정하기로 판단

- 데이터 전처리 종류

- Origin Data
- Standardization Data

$$x' = \frac{x - \bar{x}}{\sigma}$$

- Normalization Data

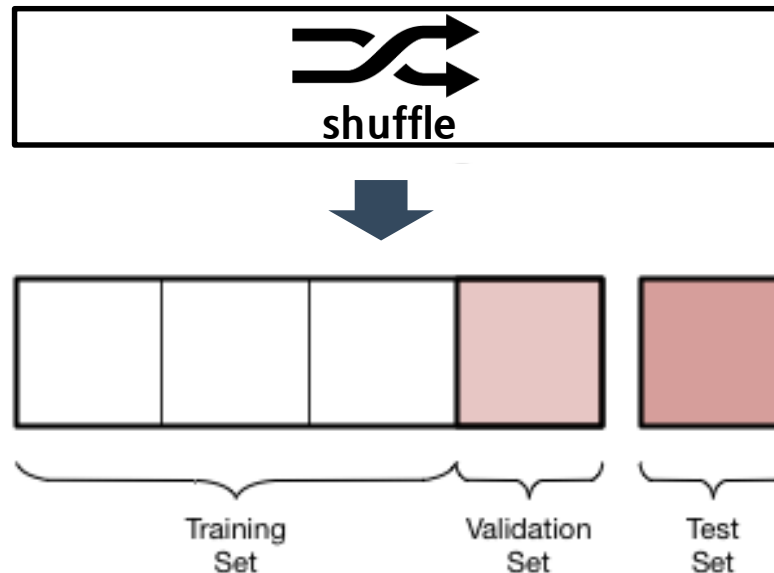
$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

## 2. 기반 모델 선정하기

### 2.3 학습 데이터 셋 / 테스트 데이터 셋 분리

- WHY?

- 앞으로 진행될 모든 모델의 학습 데이터 셋과 테스트 셋이 동일해야 동등 비교가 되기 때문에 row별로 key를 부여하여 각 Input 종류별로 동일한 데이터를 구성
- 학습 데이터 셋
  - 전체 데이터의 80%
- 테스트 데이터 셋
  - 전체 데이터의 20%
- 데이터 셋 자체가 506개로 매우 적은 데이터이기 때문에 상위 기준으로 결정



## 2. 기반 모델 선정하기

### 2.4 POC 모델 만들기

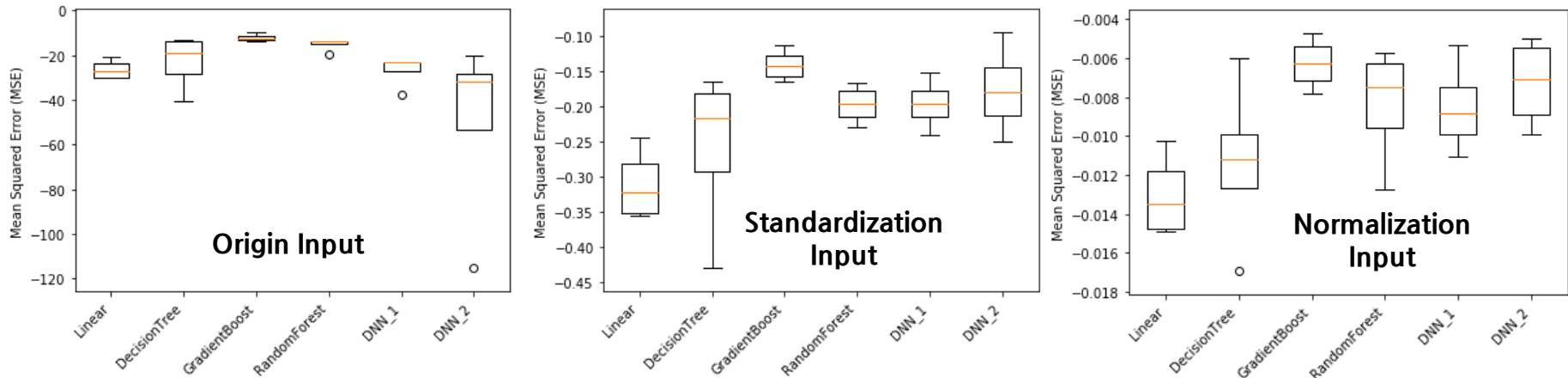
---

- WHY?
  - 여러 모델 학습 결과를 통해 어떤 종류의 데이터를 Input Feature로 선정할지 기준을 찾기 위함
  - 여러 모델 학습 결과를 통해 모델 선택 및 개선점 기준을 잡기 위함
- 모델 종류
  - Linear Regression
  - Decision Tree Regression
  - Gradient Boost Regression
  - Random Forest Regression
  - Deep Neural Network Regression
- 학습 방식
  - K-Fold Cross Validation 수행
  - Sklearn의 내장 함수 사용

## 2. 기반 모델 선정하기

### 2.4 POC 모델 만들기

- 모델 학습 결과 분석



- Origin Data를 Input으로 사용하게 되면 Feature별 편차가 크기 때문에 학습이 어려워짐
- 이에 대한 대안
  - Standardization or Normalization
- 해당 대안을 Feature에 따라 섞어서 사용할 수 있지만 한정된 작업 시간으로 인해 각각 진행
- 해당 대안의 단점
  - 학습 데이터 범위 밖의 데이터를 예측 데이터로 사용할 시 재 분석 및 재 학습이 필요
- Normalization Input POC 결과의 편차가 더 적음을 확인
- 추가적으로 Input Feature에 이미 비율에 대한 정보들이 50%를 차지하고 있어 **Normalization Input을 사용하기로 결정**
- 5개의 모델 중 성능이 좋은 **Gradient Boot, DNN**모델을 1차적으로 선정

### 3. POC 모델 개선하기

#### 목적

---

Gradient Boost 모델과 DNN 모델을

간소화한 개선을 통해

두 모델 중 성능 개선 **가능성**이 커 보이는

모델을 선택하기 위함

# 3. POC 모델 개선하기

## 3.1 EDA

- WHY?
  - 모델 개선을 위해 집값을 결정짓는 주요한 Feature를 선정하기 위함
- Feature와 집값간의 상관 관계 분석
  - 피어슨 상관 계수가 높은 Feature 분석
    - RM(방 수)의 경우 집 가격과 매우 밀접한 관련이 있음 (양의 상관 관계)
    - PTRATIO(학생 교사 비율)의 경우 교사 한 명당 학생 비율이 많으면 집 값이 떨어짐 (음의 상관 관계)
    - LSTAT(하층민 비율)의 경우 비율이 많을 수록 집 값이 떨어짐(음의 상관 관계)
  - Skew Data와의 상관 계수 분석
    - CRIM(범죄율)이 0에 가까운 데이터를 제거 후 상관 관계를 보면 범죄율에 따라 집값이 음의 상관 관계를 보이는 것을 알 수 있음
    - B(흑인 비율)는 집 값에 영향을 주지 않는 것으로 보임(대부분의 데이터가 비슷한 값에 분포 됨)
    - ZN(지대가 높은 곳의 비율)의 경우 집값에 영향을 주지 않는 것으로 보임



# 3. POC 모델 개선하기

## 3.1 EDA

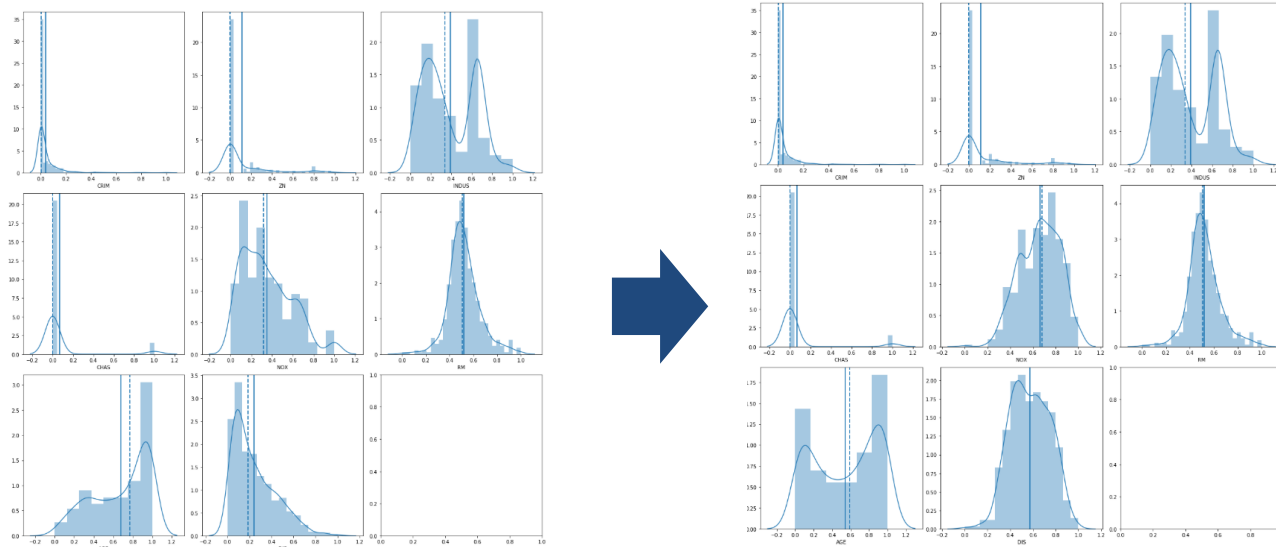
- 나머지 Feature 분석
  - AGE(오래된 주택 비율)의 경우 오래된 주택의 비율이 많을 수록 집 값이 떨어지는 음의 상관 관계의 경향이 보임.
  - DIS(직원 센터와의 접근성)의 경우 접근성이 좋을 수록 집 값이 올라가는 양의 상관 관계의 경향을 보임
  - NOX(질소 비율)의 경우 집값과 음의 상관 관계의 경향성을 보임.
  - INDUS(비 상업지역 비율)의 경우 집값과 음의 상관 관계의 경향성을 보임.
  - TAX(재산세)의 경우 집 값과 상관 관계를 보인다고 보기 어려움.
  - RAD(고속도로 접근성)의 경우 집 값과 상관 관계를 보인다고 보기 어려움.
- Gradient Boost Feature Importance 분석
  - 2 Chapter에서 진행한 모델을 통해 Feature Importance를 추출한 결과
  - 상위 TOP5 - RM, DIS, LSTAT, AGE, CRIM
- 최종 결정 Feature 8개
  - RM, LSTAT, PTRATIO, CRIM, INDUS, NOX, AGE, DIS

# 3. POC 모델 개선하기

## 3.2 Feature Engineering

- WHY?

- 최종 결정된 Feature를 회귀분석 표준 가정에 맞추기 위해 정규 분포화 유사한 형태로 변형하기 위함
- 사전에 생성한 Normalization Feature와 변형한 Feature간의 성능 비교 목적
- 변형이 가능한 Feature만 선별적으로 수행
- 극단적으로 Skew된 데이터는 해당 변환에서 제외(데이터의 의미를 왜곡하기 때문이라 판단)
  - Skew 데이터의 경우 Categorical Encode data로 변형하여 튜닝 할 예정

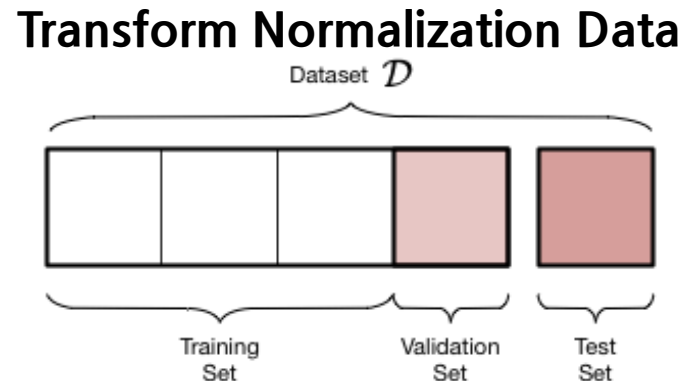
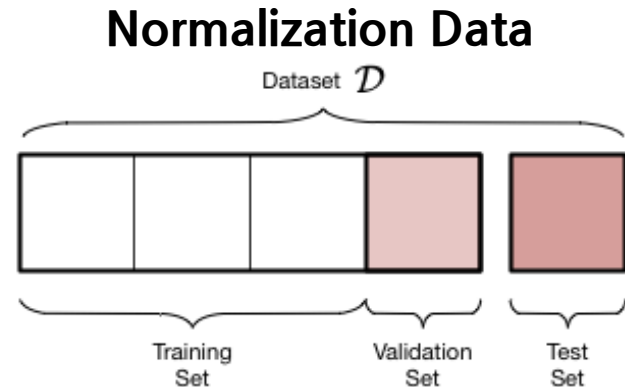
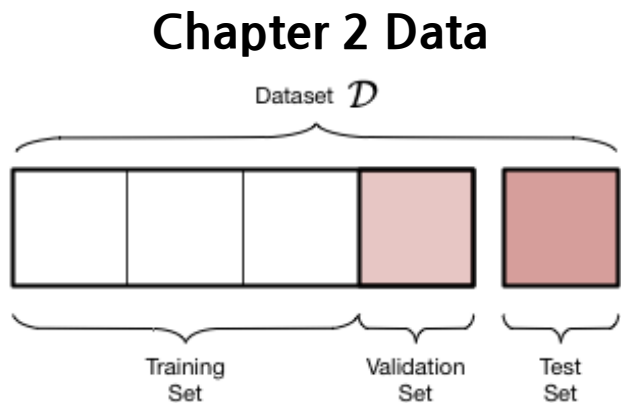


# 3. POC 모델 개선하기

## 3.3 학습 데이터 셋 / 테스트 데이터 셋 분리

- WHY?

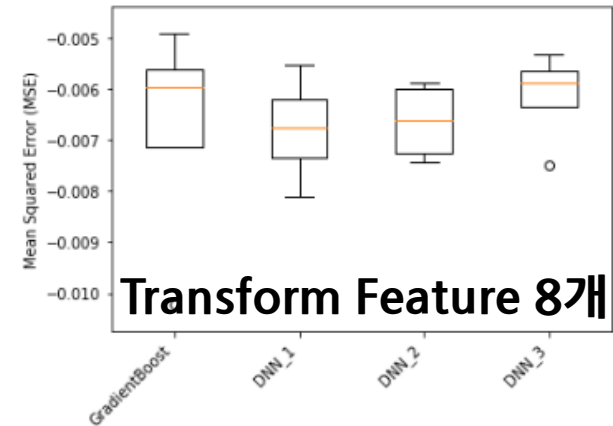
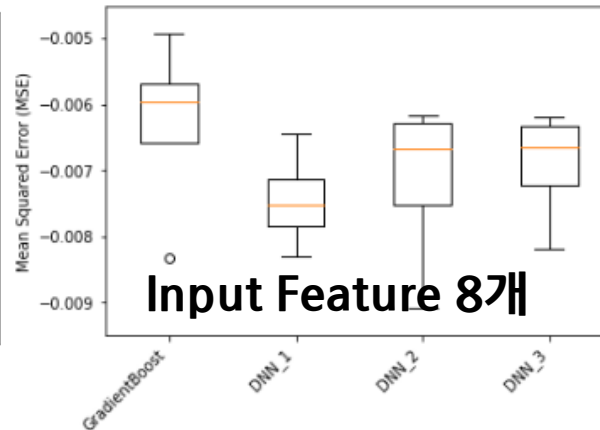
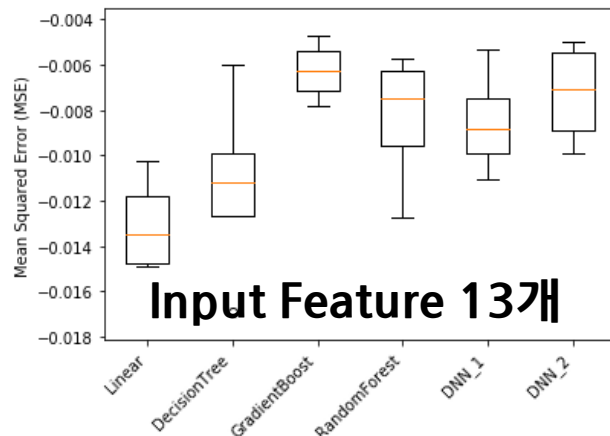
- 상위 데이터 셋과 동일 기준으로 Normalization Feature와 정규 분포로 부분 변환한 Normalization Feature의 테스트 셋 테스트 데이터 셋을 분리



# 3. POC 모델 개선하기

## 3.4 Input별 성능 비교

- WHY?
  - Normalization Feature와 Transform Normalization Feature간 성능 비교
- 모델 학습 결과 분석



- Feature Selection 결과
  - 기존 Feature 13개를 Input으로 학습 했을 때보다 8개의 Feature를 사용했을 때 성능이 개선됨을 확인
- Feature Transform 결과
  - Feature의 분포를 변형한 결과가 변경 전 보다 더 좋은 결과를 얻음(DNN 모델 한정)
- 8개 Feature와 Transform Feature를 사용하여 모델을 튜닝을하기로 결정

## 3. POC 모델 개선하기

### 3.5 모델 튜닝

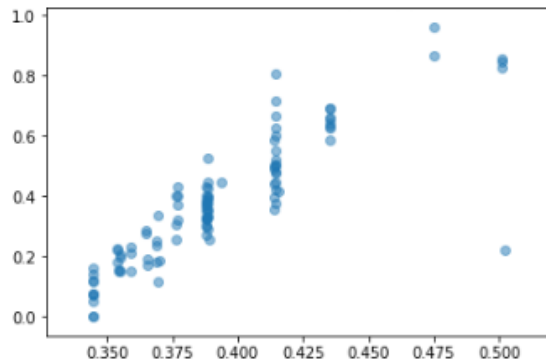
- WHY?
  - 두 모델의 간소화 튜닝 작업을 통해 최종 Deep Dive할 모델을 선정하기 위함
- K-fold를 사용하지 않고 전체 데이터를 학습하여 두 모델간의 예측 값에 따른 결과 비교를 수행 (빠른 결과 확인 목적)
- 동일 Feature 동일 Train, Test Set을 이용하여 학습 진행
- 결과 분석
  - 전반적으로 DNN 모델이 튜닝에 따라 결과가 조금 더 개선되는 것으로 확인됨
  - Gradient Boost 모델의 경우 개선의 한계가 보임 (확신하기 힘들, 더 많은 테스트 필요)
  - DNN 모델의 경우 Optimizer 변경에 따라 성능이 개선 됨 (Adam이 RMSprop보다 더 좋은 결과를 냄)
  - DNN 모델의 경우 Drop Out 추가에 따라 네트워크 표현력이 증가하게 되어 과적합 이슈를 개선함
  - DNN 모델의 경우 Regularization 추가에 따라 과적합 이슈를 개선함
  - 해당 모델 환경에서 최적의 Epoch은 600정도로 보임 (Deep Dive 시 변경 가능성 있음)
  - 간소한 모델 튜닝 작업을 통해 얻은 DNN모델의 최종 Mean Square Error는 **0.004732**
  - **DNN 모델의 성능을 끌어 올리기 위해 조금 더 Deep Dive 하기로 결정**

# 3. POC 모델 개선하기

## 3.5 모델 튜닝(과정 별 Metric)

- Gradient Boost 모델 튜닝 작업

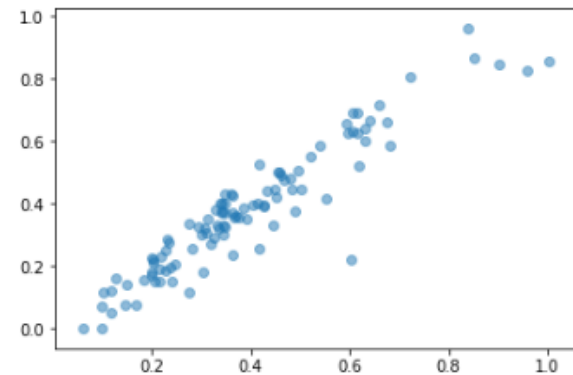
[INFO] MSE : 0.030624



Learning Rate 변경



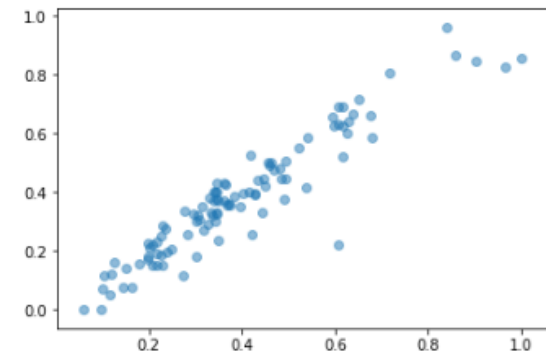
[INFO] MSE : 0.008093



n estimator 변경



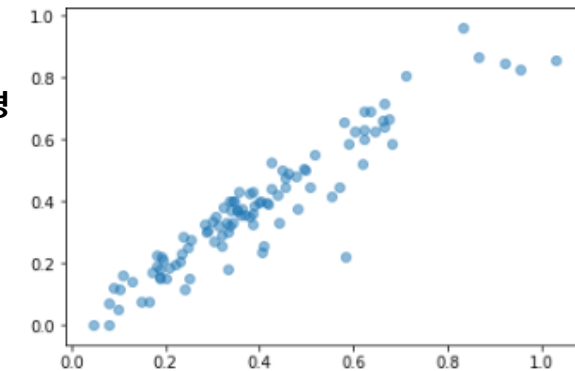
[INFO] MSE : 0.008002



n estimator 변경  
Learning Rate 변경



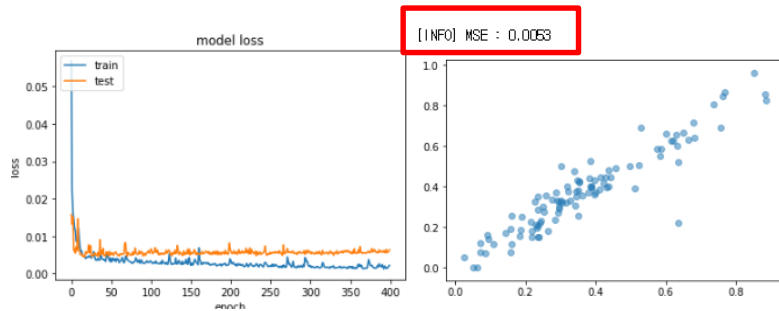
[INFO] MSE : 0.008008



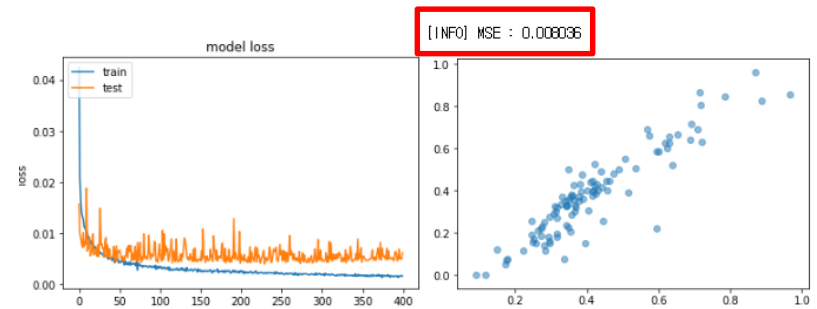
# 3. POC 모델 개선하기

## 3.5 모델 튜닝(과정 별 Metric)

- DNN 모델 튜닝 작업

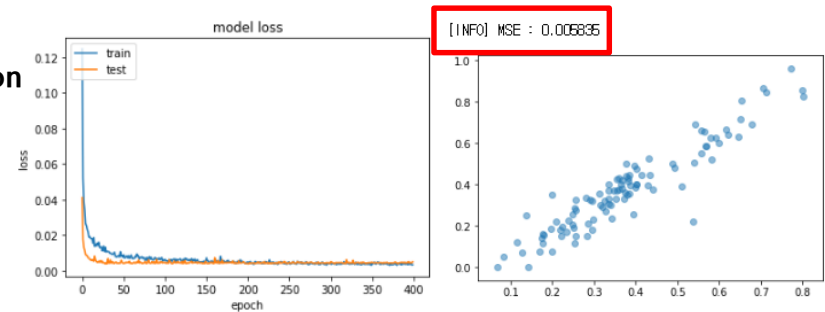
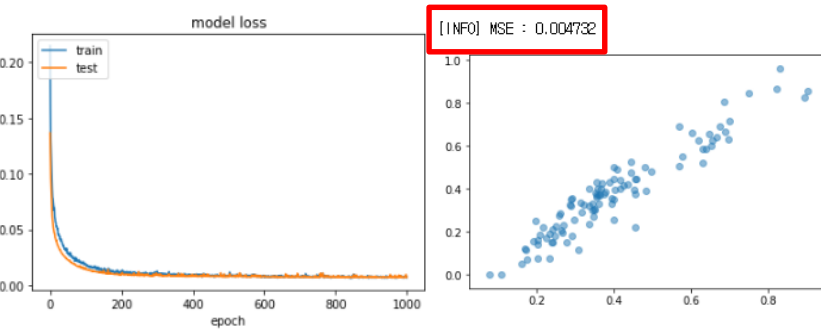


Optimizer 변경



Drop Out 추가  
Optimizer 원복

Regularization  
추가



## 4. 최종 모델 Deep Dive

### 목적

---

집값 예측에 가장 적합한

**모델**을 만들기 위함



## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- WHY?
  - Network Layer를 깊게 쌓아 표현력을 강화하기 위함

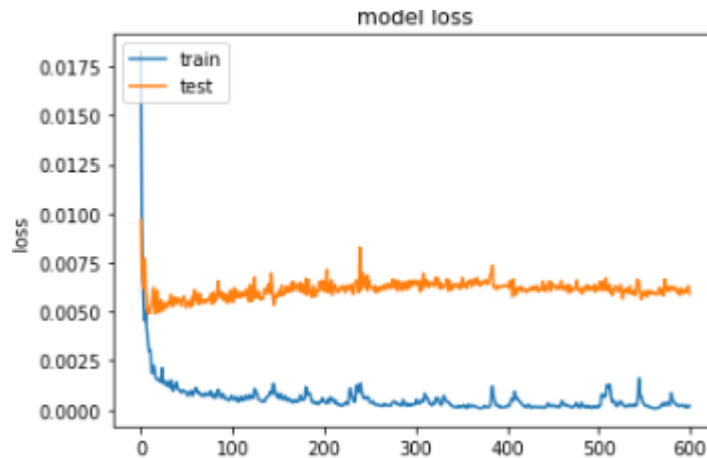
Layer (type)	Output Shape	Param #
dense_22 (Dense)	(None, 64)	576
dense_23 (Dense)	(None, 64)	4160
dense_24 (Dense)	(None, 64)	4160
dense_25 (Dense)	(None, 64)	4160
dense_26 (Dense)	(None, 64)	4160
dense_27 (Dense)	(None, 1)	65
Total params: 17,281		
Trainable params: 17,281		
Non-trainable params: 0		

- K-Fold Cross Validation 수행
- Sklearn의 내장 함수를 사용하지 않음
  - 각 Fold의 Learning Curve History 평균 확인 목적

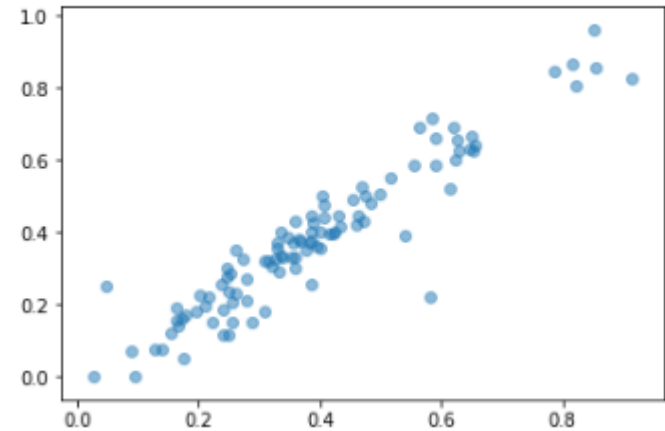
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 1
  - 목적 : Base Line 생성 목적
- 내용
  - 5 Hidden Layer 기본 모델 학습
- 결과



[INFO] MSE : 0.004887

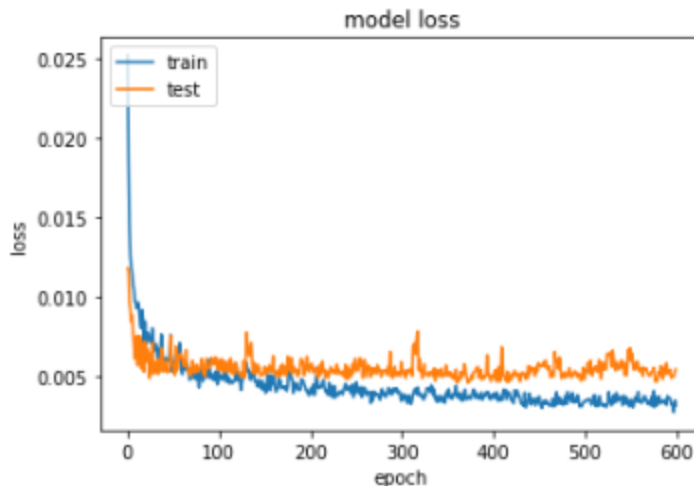


- 과적합 이슈가 매우 극명하게 나타남

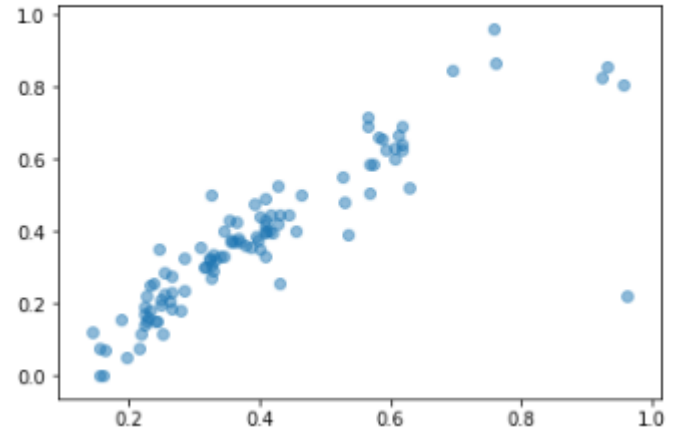
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 2
  - 목적 : Drop Out 추가에 따른 변화 확인
- 내용
  - Drop Out Node 비유 20% 설정
- 결과



[INFO] MSE : 0.010854

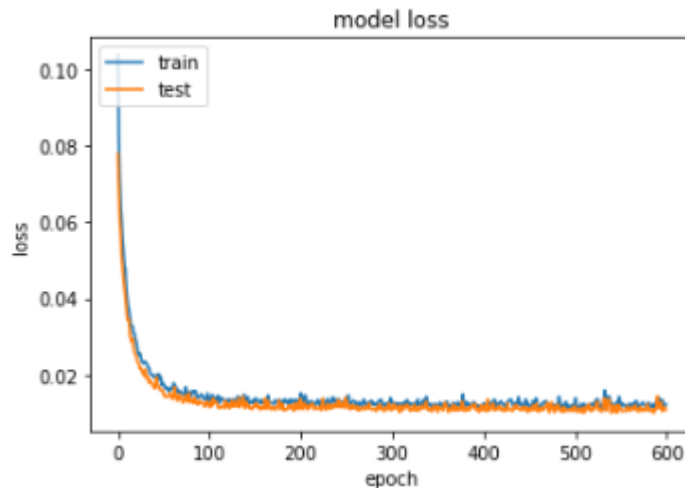


- 네트워크 과적합 이슈가 감소됨을 확인
- 하지만 여전히 과적합 이슈는 존재
- MSE는 더 커지는 문제 발생
- Drop Out Node 비율이 너무 높아 오히려 표현력을 잃은 것이 아닌가라는 판단이 들

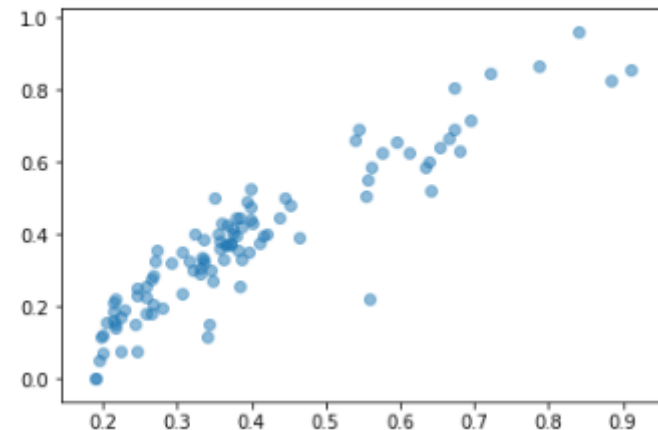
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 3
  - 목적 : Regularization 추가에 따른 변화 확인
- 내용
  - L2 Regularization 추가
- 결과



[INFO] MSE : 0.006948

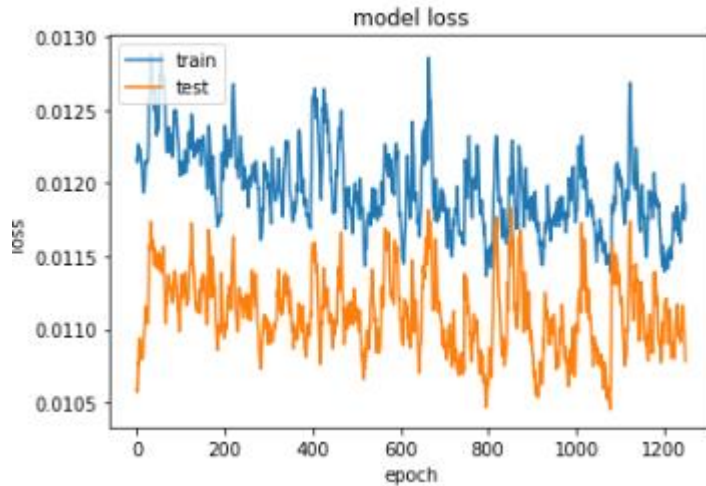


- Drop Out 비율 조정 (20% -> 10%) , L2 Regularization 추가를 통해 과적합 이슈를 많이 해결할 수 있었음. 일반화가 잘되었다 라는 판단이 됨
- MSE가 그래도 높은 이슈는 존재
- Epoch을 늘려 학습시간을 길게 잡으면 조금 더 Fitting된 결과를 얻을 수 있지 않을까 판단이 듦

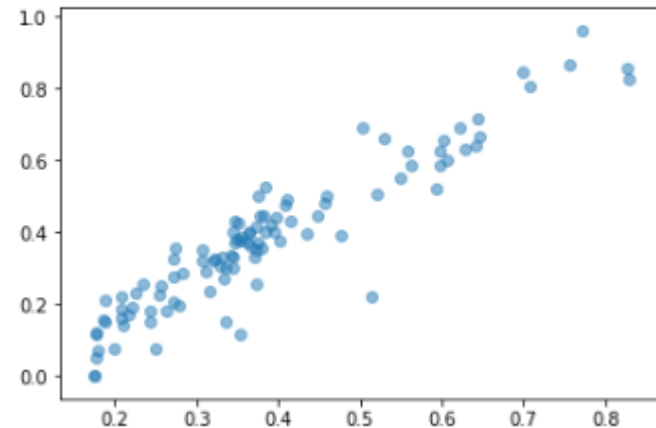
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 4
  - 목적 : 조금 더 Fitting된 결과를 보기 위해서 학습 시간을 늘림.
- 내용
  - Epoch 600 -> 1500
- 결과



[INFO] MSE : 0.006442

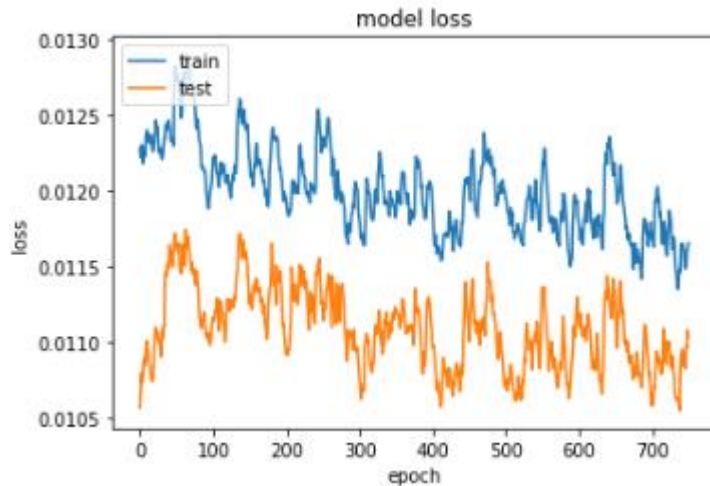


- 학습시간을 늘리니 MSE가 조금 더 감소하는 것을 확인
- Network Optimize가 필요하다고 판단

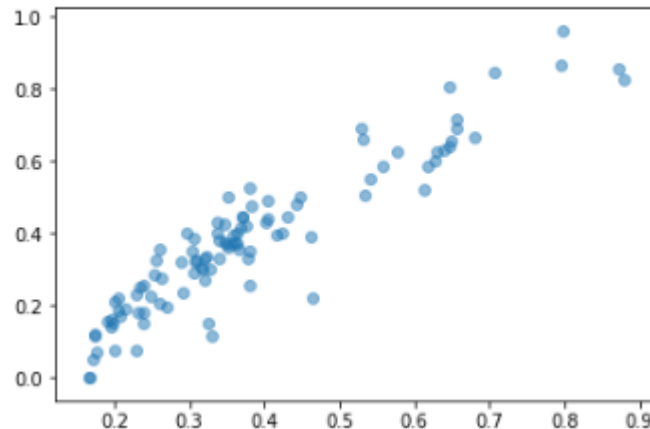
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 5
  - 목적 : Network Optimize를 위해 가중치 초기화 기법 변환
- 내용
  - Random → Xavier
- 결과



[INFO] MSE : 0.006014

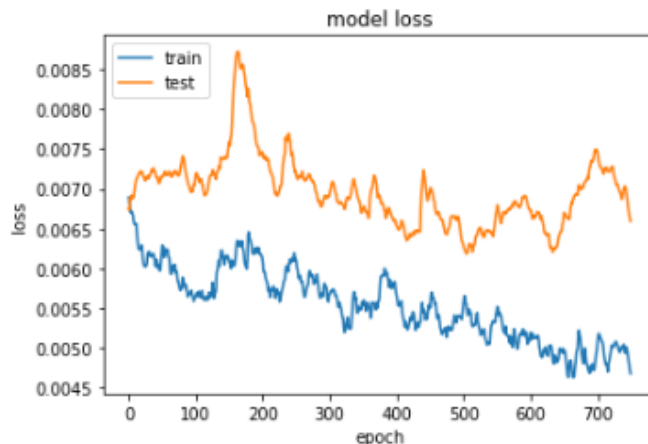


- Xavier initialize를 추가하여 MSE가 감소함을 확인
  - ReLU Activation Function을 사용하기에 He initialize가 더 성능이 좋을 것이라 판단했지만 Xavier가 더 좋은 결과를 얻음
- 성능향상 목적 추가적인 Network Optimize 수행이 필요하다 판단

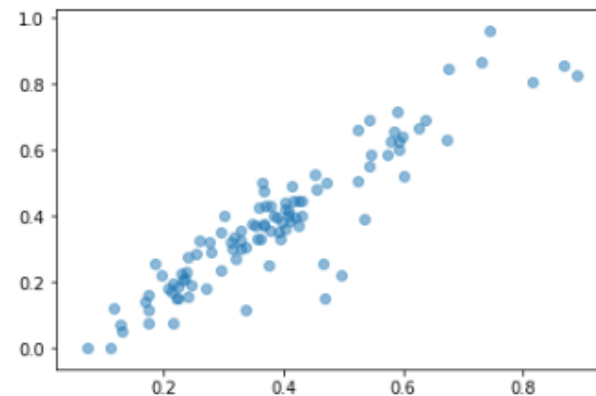
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 6
  - 목적 : Network Optimize를 위해 Batch Normalization 추가
- 내용
  - 모든 Hidden Layer에 Batch Normalization 기능 추가
  - Drop Out Regularization 제거 후 Batch Normalization 적용
  - 학습시간 증가 및 Learning Rate 변경
- 결과



[INFO] MSE : 0.006801

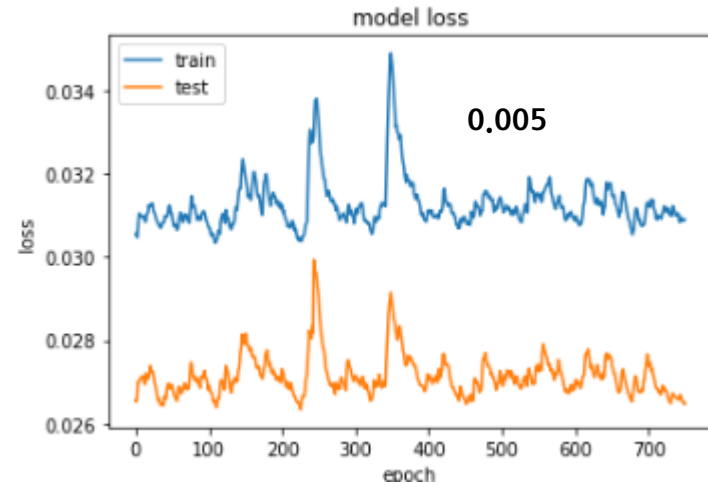
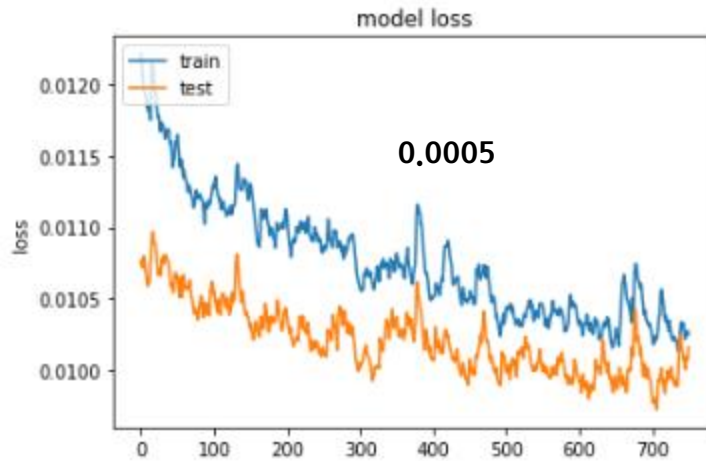


- Batch Normalization을 통한 성능 향상은 얻지 못함. 오히려 성능이 많이 떨어짐을 확인
- Input 자체가 Normalization된 데이터이기 때문에 Batch Normalization 효과를 이미 반영하고 있던 것이 아닌가라는 판단이 듦
- 또한 과적합 이슈 재 발생

## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 7
  - 목적 : Network Optimize를 위해 Learning Rate 변경
- 내용
  - Learning Rate - 0.001  $\rightarrow$  0.0005, 0.001  $\rightarrow$  0.005
- 결과



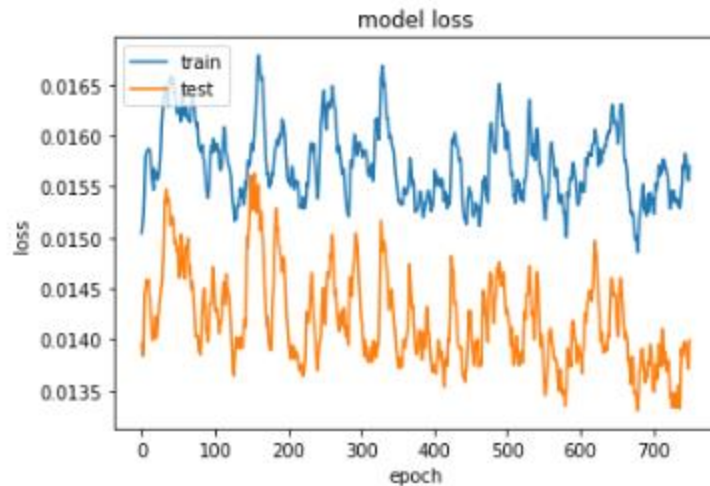
- Learning rate 감소에 따른 성능향상은 없었음
- Learning rate 증가에 따라 성능이 저하됨
- 해당 Input과 네트워크에 가장 적합한 Learning rate는 0.001이라고 판단됨



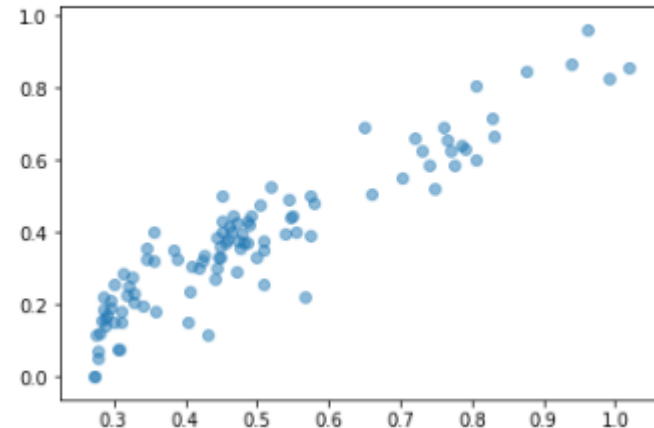
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 8
  - 목적 : Network Optimize를 위해 Mini Batch Size 변경
- 내용
  - Mini Batch Size 16 → 32, 16 → 64
- 결과



[INFO] MSE : 0.018463

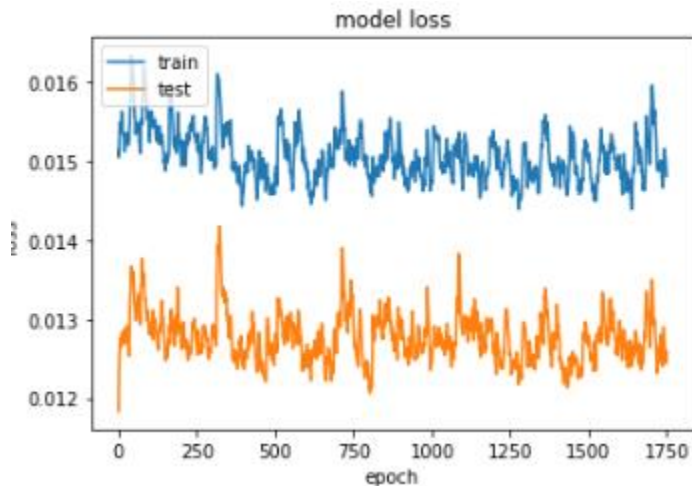


- Mini Batch Size 조절에 따라 오히려 성능이 저하됨을 확인

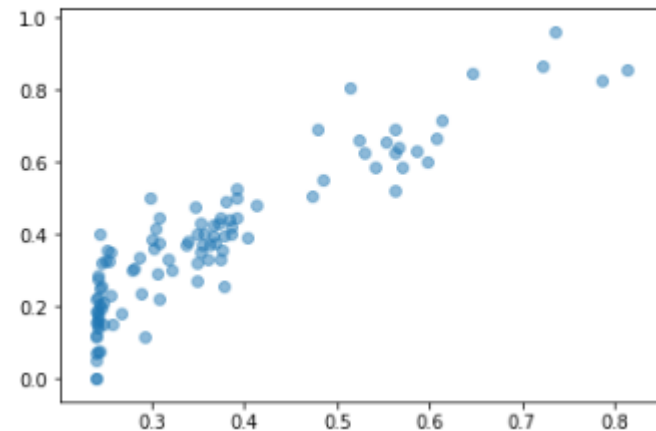
## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 실험 9
  - 목적 : 상위 결과를 토대로 가장 적합한 Epoch을 얻기 위함
- 내용
  - Epoch 1000 -> 2000
- 결과



[INFO] MSE : 0.009861

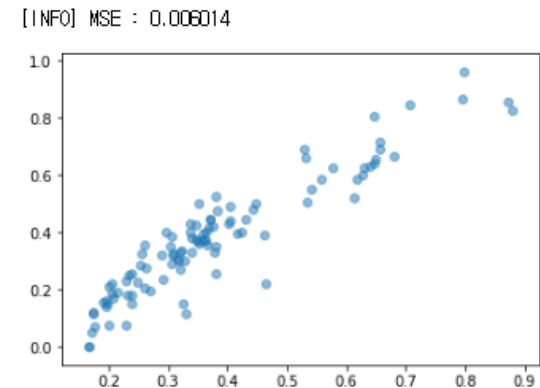


- 학습 시간이 길어진다 해도 Loss는 더 이상 하락하지 않음

## 4. 최종 모델 Deep Dive

### 4.1 Hidden Layer 증가에 따른 모델 튜닝

- 결과 분석
  - 5 Hidden Layer 모델에서 가장 성능이 높은 환경은 아래와 같음
  - Epoch 1000
  - Mini Batch Size 16
  - Learning Rate 0.001
  - Adam Optimizer
  - Regularization
  - Drop Out
  - Xavier Init
  - 최종 평가 데이터 셋에 대한 최소 MSE - **0.006014**



- 최종 결과가 Chapter3에서 진행했던 2Hidden Layer보다 좋은 성능을 보이지 못함
- 본 Data는 2 Hidden Layer 모델이 적합한 모델이 아니라는 판단이 들

## 4. 최종 모델 Deep Dive

### 4.2 Hidden Layer 감소에 따른 모델 튜닝

- WHY?
  - 실험중인 모델 중 가장 좋은 성능을 보인 모델을 개선하기 위함

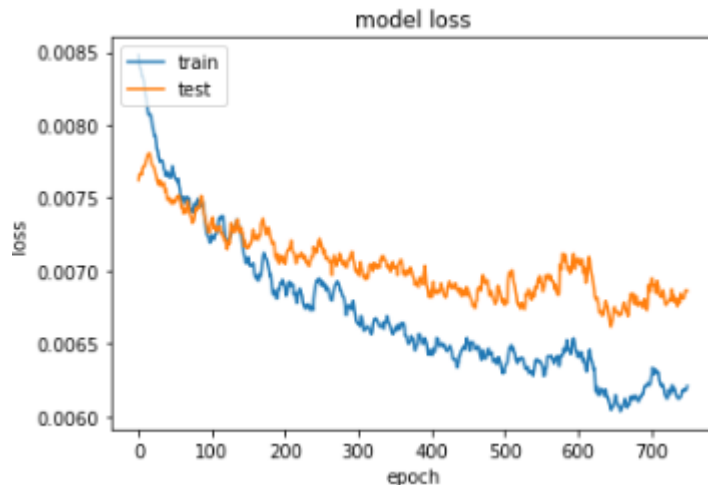
Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 64)	576
dropout_15 (Dropout)	(None, 64)	0
dense_29 (Dense)	(None, 64)	4160
dropout_16 (Dropout)	(None, 64)	0
dense_30 (Dense)	(None, 1)	65
Total params: 4,801		
Trainable params: 4,801		
Non-trainable params: 0		

- K-Fold Cross Validation 수행
- Sklearn의 내장 함수를 사용하지 않음
  - 각 Fold의 Learning Curve History 평균 확인 목적

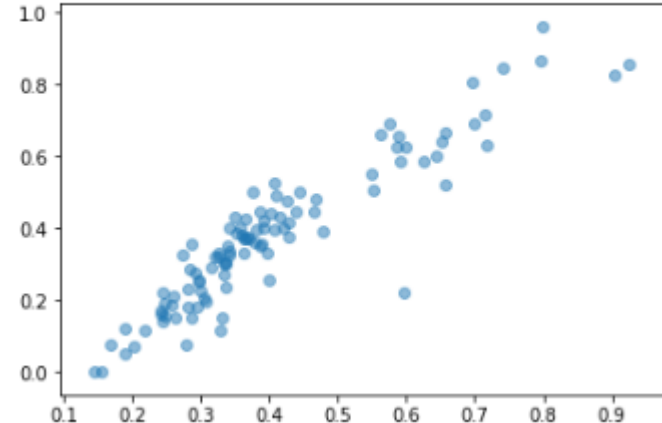
## 4. 최종 모델 Deep Dive

### 4.2 Hidden Layer 감소에 따른 모델 튜닝

- 실험 1
  - 2 Hidden Layer의 Base Line을 잡기 위함
- 내용
  - 상위 4.1 실험과 유사한 Option을 적용
  - L2 Regularization 추가
- 결과



[INFO] MSE : 0.007419

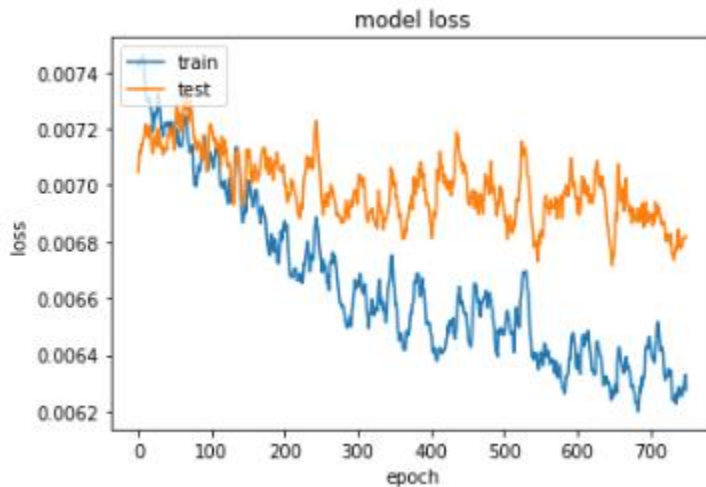


- Loss는 전반적으로 5 hidden layer보다 낮음을 확인
- 하지만 과적합 이슈 존재

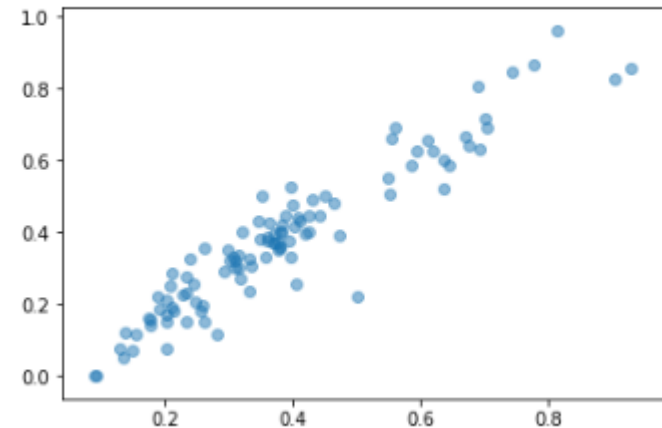
## 4. 최종 모델 Deep Dive

### 4.2 Hidden Layer 감소에 따른 모델 튜닝

- 실험 2
  - 네트워크 성능 개선 목적
- 내용
  - Xavier Init 추가
- 결과



[INFO] MSE : 0.004676

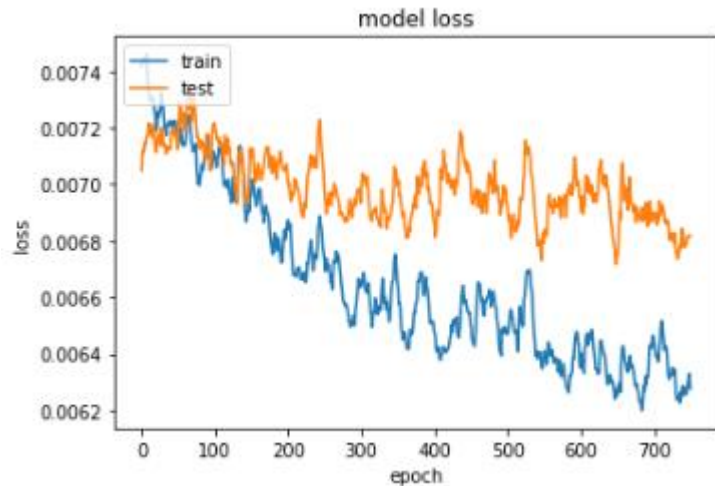


- Loss는 전반적으로 이전 실험 보다 개선됨을 확인
- 하지만 과적합 이슈는 그래도 존재

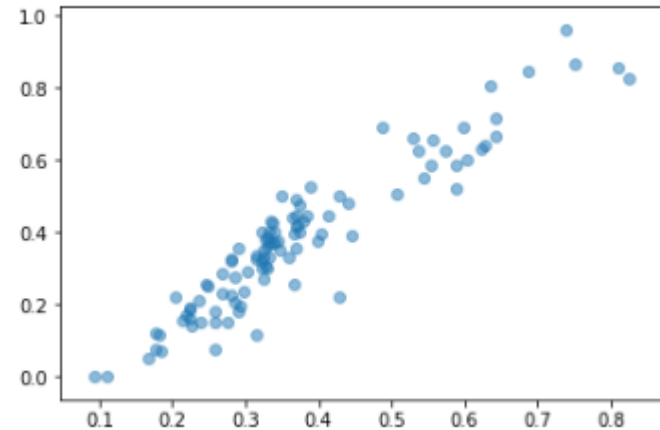
## 4. 최종 모델 Deep Dive

### 4.2 Hidden Layer 감소에 따른 모델 튜닝

- 실험 3
  - 네트워크 성능 개선 목적
- 내용
  - Learning Rate 변경 0.001 -> 0.0008
- 결과



[INFO] MSE : 0.006592

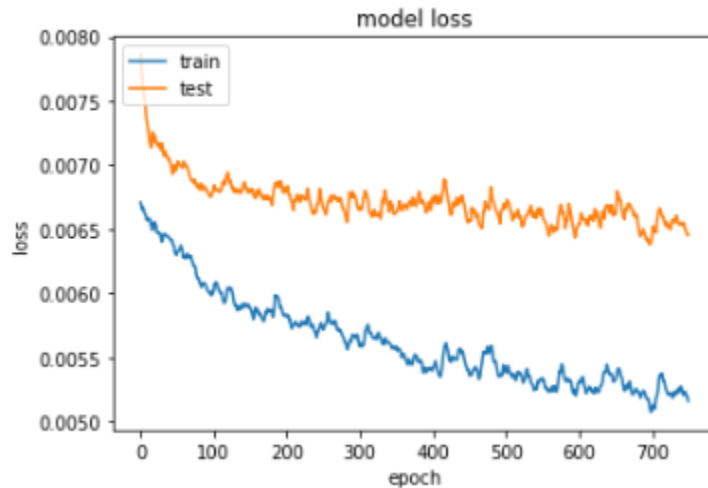


- 과적합 이슈는 그대로 존재
- Learning rate 변경에 따라 성능이 개선되지 않음
- 기존 설정이 더 맞는 설정이라 판단됨

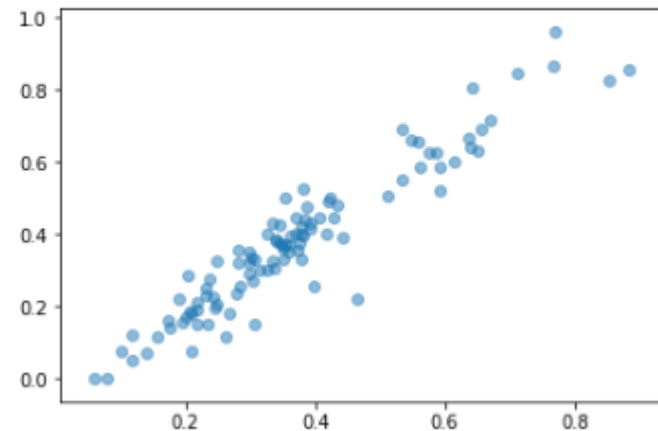
## 4. 최종 모델 Deep Dive

### 4.2 Hidden Layer 감소에 따른 모델 튜닝

- 실험 4
  - 과적합 문제 해결 목적
- 내용
  - Drop Out 비율 변화 0.2  $\rightarrow$  0.1
- 결과



[INFO] MSE : 0.004886



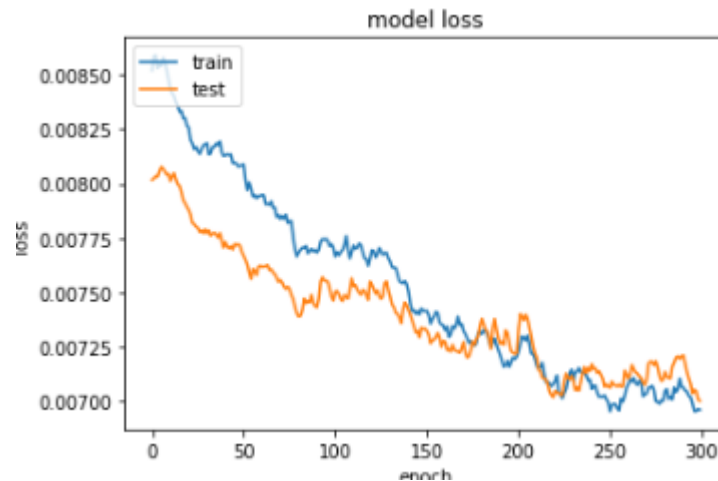
- Loss가 많이 이전 실험 대비 많이 감소하는 것을 확인
- 과적합 이슈는 그대로 존재함



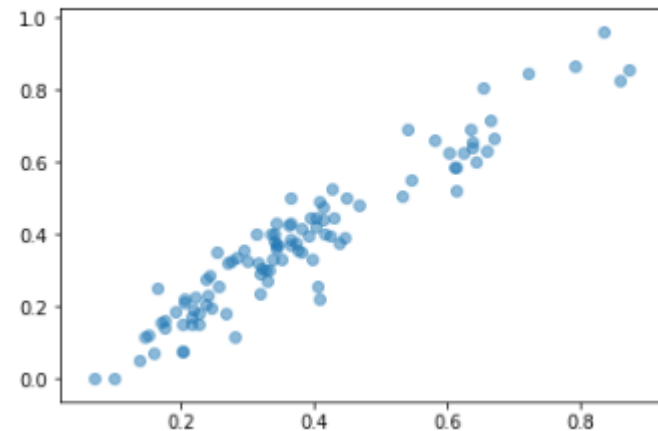
## 4. 최종 모델 Deep Dive

### 4.2 Hidden Layer 감소에 따른 모델 튜닝

- 실험 5
  - 과적합 문제 해결 목적
- 내용
  - Early Stop 1000 → 500
- 결과



[INFO] MSE : 0.004276



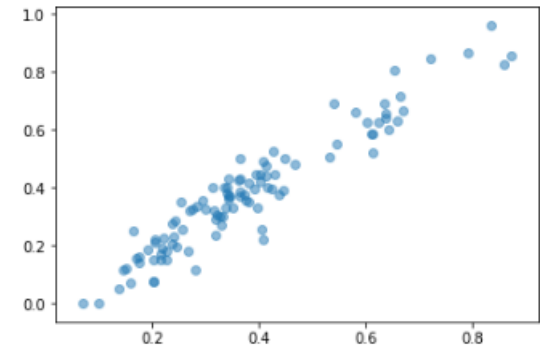
- Early Stop을 통해 최적지점의 학습시간 도출
- 과적합 이슈 개선
- 적절한 일반화 수행

# 4. 최종 모델 Deep Dive

## 4.2 Hidden Layer 감소에 따른 모델 튜닝

- 결과 분석
  - 2 Hidden Layer 모델에서 가장 성능이 높은 환경은 아래와 같음
  - Epoch 500
  - Mini Batch Size 32
  - Learning Rate 0.001
  - Adam Optimizer
  - Regularization
  - Drop Out
  - Xavier Init
  - 최종 평가 데이터 셋에 대한 최소 MSE - **0.004276**
- 2 Hidden Layer 모델은 가장 성능이 좋은 모델로 보임
- 해당 모델은 추후 성능 평가 및 비교를 위해 별도 저장

[INFO] MSE : 0.004276

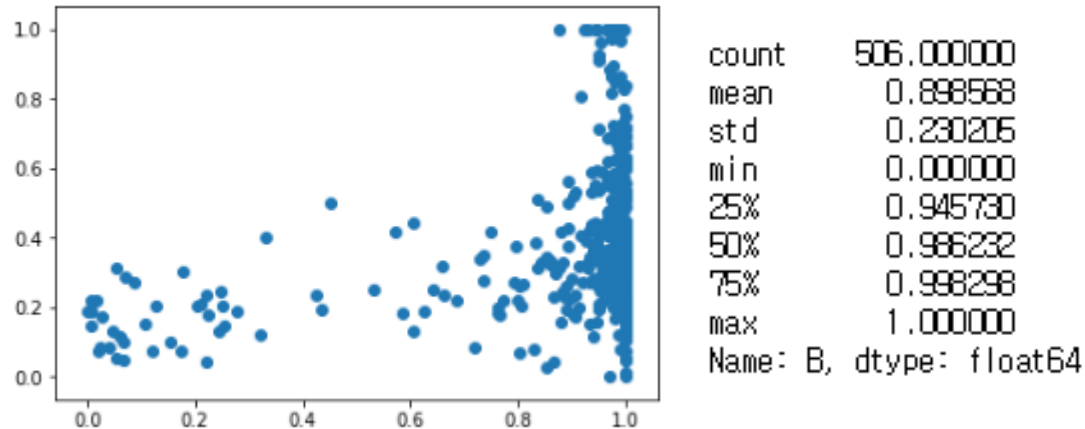


teriusbin Add files via upload ...			Latest commit 1d7a701 18 hours ago
README.md	Initial commit		18 hours ago
attention_model.h5	Add files via upload		18 hours ago
attention_model.json	Add files via upload		18 hours ago
boston_house_predict.ipynb	Add files via upload		18 hours ago
model.h5	Add files via upload		18 hours ago
model.json	Add files via upload		18 hours ago
README.md			

## 4. 최종 모델 Deep Dive

### 4.3 Feature 추가 및 변경에 따른 모델 튜닝

- EDA ( B Column )

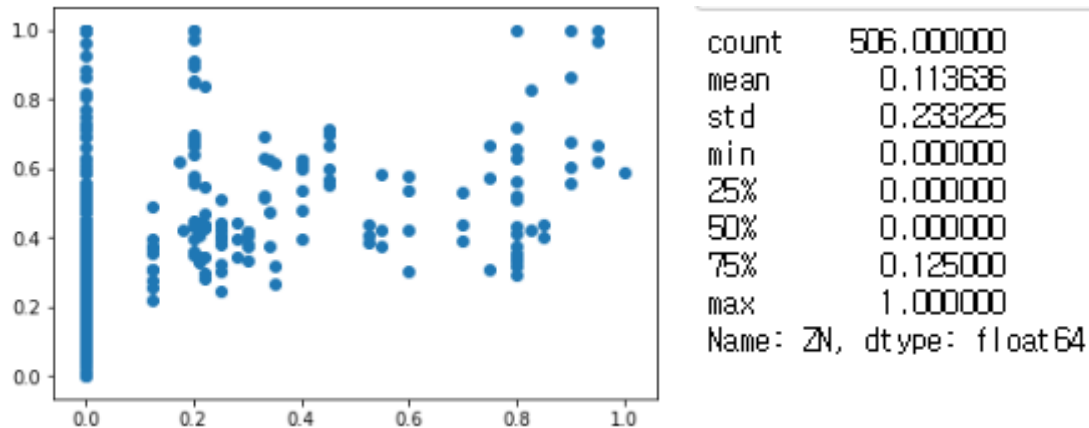


- Percentile 25%이상의 데이터가 대부분 0.94%이상의 데이터로 분포하고 있음
- B는 피어슨 계수로 확인하였을 때 TAX,RAD,DIS와 연관 관계가 깊음
- 집값과의 관계는 뚜렷하지 않지만 0.9이하의 데이터는 양의 상관 관계의 경향성이 보임

## 4. 최종 모델 Deep Dive

### 4.3 Feature 추가 및 변경에 따른 모델 튜닝

- EDA ( ZN Column )

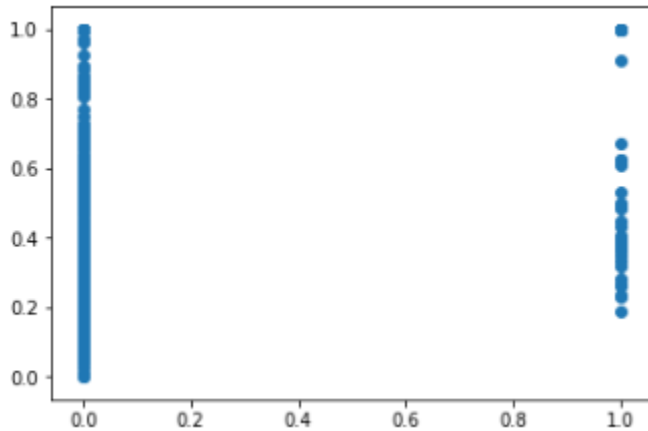


- Percentile 75% 이하의 데이터가 대부분 0.0에 분포하고 있음
- 0.0 이상인 데이터들은 많지 않지만 집값과 양의 상관 관계의 경향성을 보임
- 해당 데이터는 DIS 데이터와 피어슨 상관계수가 높음
- DIS 데이터와 함께 집값에 영향을 줄 수 있는 여지가 있어 보임

## 4. 최종 모델 Deep Dive

### 4.3 Feature 추가 및 변경에 따른 모델 튜닝

- EDA ( CAHS Column )

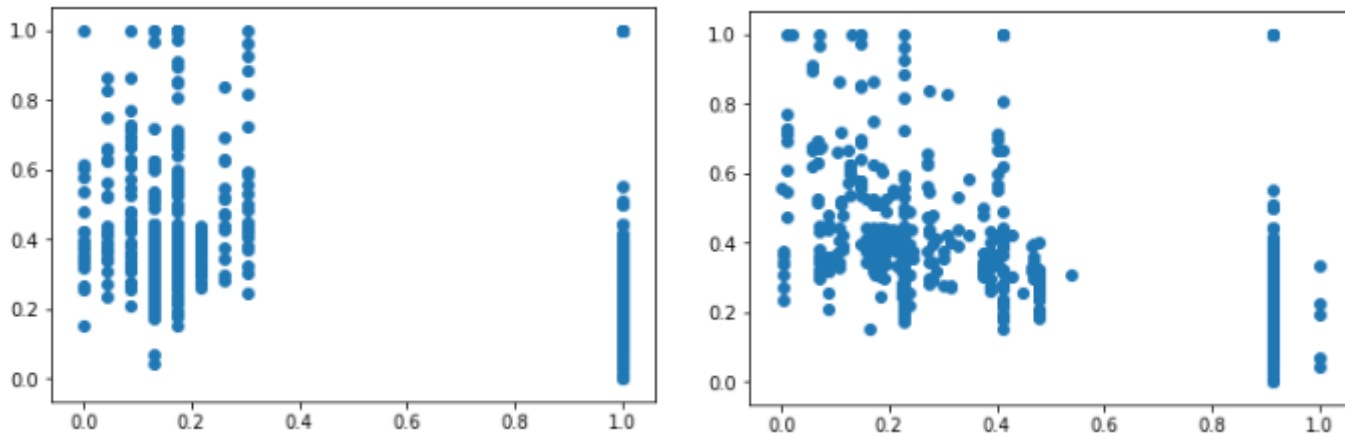


- 대부분 0으로 구성되어 있는 Categorical 데이터

## 4. 최종 모델 Deep Dive

### 4.3 Feature 추가 및 변경에 따른 모델 튜닝

- EDA ( RAD& TAX Column )

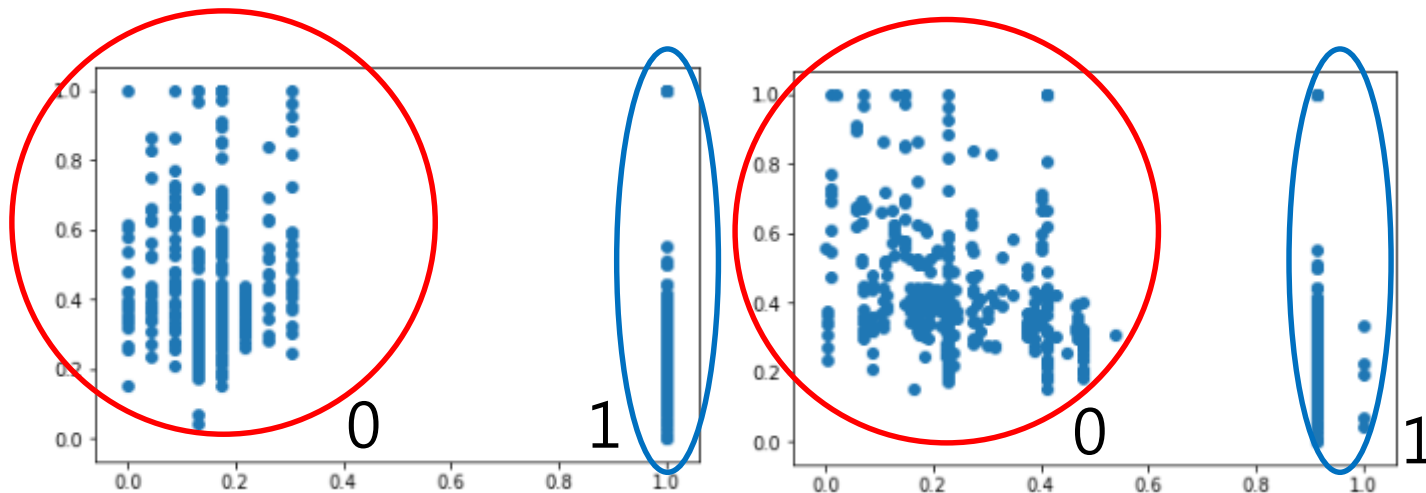


- TAX와 RAD는 매우 높은 상관 관계를 가지고 있음
- 하지만 집값과의 관계를 정의하기는 힘들
- RAD의 경우 0.4아래 데이터와 0.8 이상 데이터, TAX의 경우 0.6이하 데이터와 0.8이상 데이터를 Categorical하게 묶어서 생각했을 경우 집값과 연관성이 있을 수 있다고 판단
- 해당 데이터를 Binary Encoding 하여 Input으로 결정

## 4. 최종 모델 Deep Dive

### 4.3 Feature 추가 및 변경에 따른 모델 튜닝

- Feature Engineering ( Feature Binary Encoding )



```
X_train_all_feature = X_train_scale[['RM', 'LSTAT', 'PTRATIO', 'CRIM', 'INDUS', 'NOX', 'AGE', 'DIS', 'B', 'ZN', 'CHAS', 'RAD',  
    'TAX', 'key']]  
Y_train_all_feature = Y_train_scale[['MEDV', 'key']]  
X_test_all_feature = X_test_scale[['RM', 'LSTAT', 'PTRATIO', 'CRIM', 'INDUS', 'NOX', 'AGE', 'DIS', 'B', 'ZN', 'CHAS', 'RAD', 'TA  
X', 'key']]  
Y_test_all_feature = Y_test_scale[['MEDV', 'key']]
```

```
X_train_all_feature['TAX'] = X_train_all_feature.TAX.apply(lambda x: 1 if x > 0.6 else 0)
```

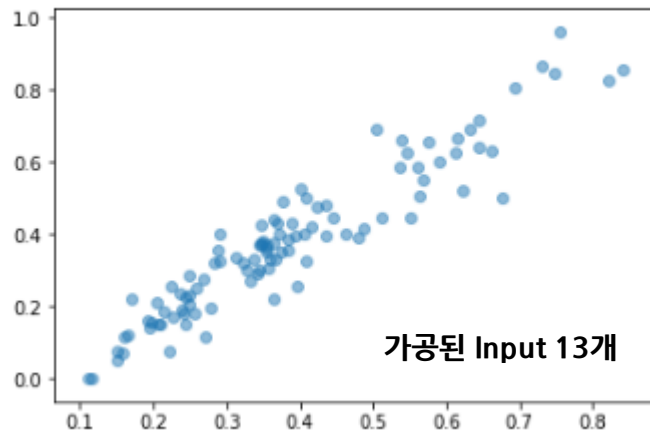
```
X_train_all_feature['RAD'] = X_train_all_feature.TAX.apply(lambda x: 1 if x > 0.4 else 0)
```

## 4. 최종 모델 Deep Dive

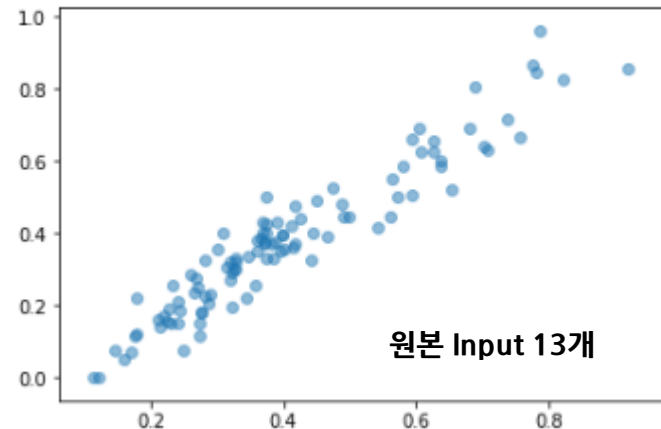
### 4.3 Feature 추가 및 변경에 따른 모델 튜닝

- 실험
  - 목적 : 원본 Input Feature 13개를 모두 사용했을 경우와 가공한 Input Feature 13개를 사용했을 때의 성능 비교
- 결과

[INFO] MSE : 0.006262



[INFO] MSE : 0.004842



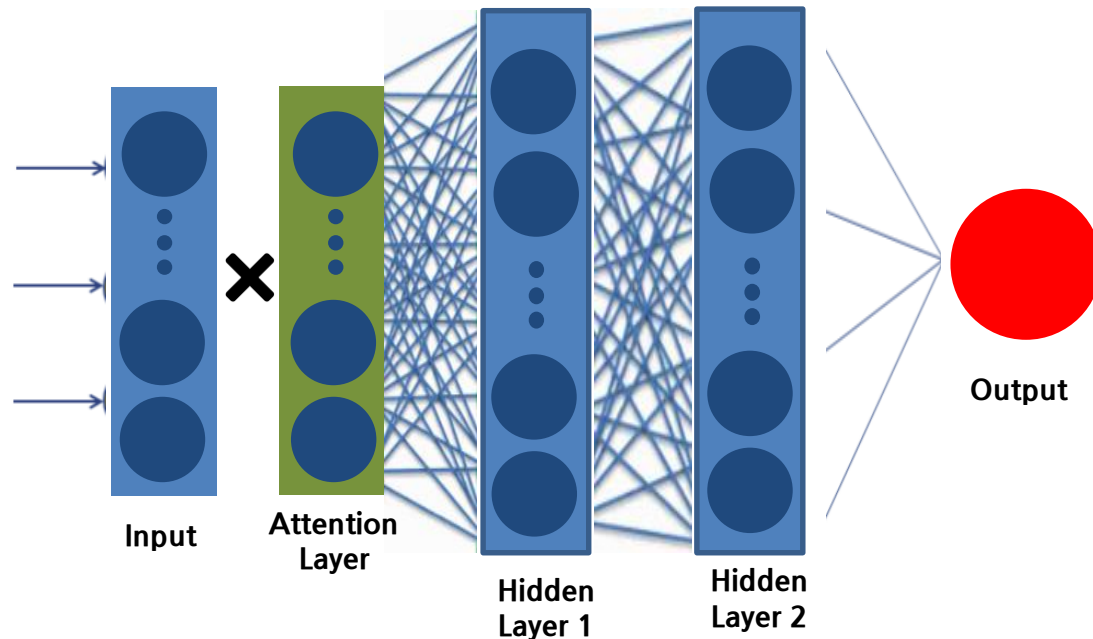
- 최종적으로 Feature Selection( 8개의 Feature ) 결과보다 낮은 성능을 보임
- 범주형 인코딩 Input을 사용했을 때보다 원본 Input을 사용했을 때의 결과가 조금 더 좋은 표현력을 가지는 것으로 판단됨



## 4. 최종 모델 Deep Dive

### 4.3 Attention Mechanism 기반 Feature Importance 측정

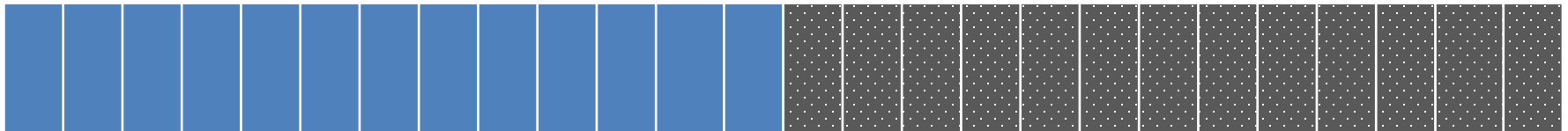
- WHY?
  - 지금까지 가장 최선이라고 생각하고 있는 모델의 Feature가 정말로 잘 뽑힌 Feature인가를 확인하기 위함
- 내용
  - 첫 번째 DNN Layer에 Attention Layer를 달고 해당 네트워크가 Regression을 잘 하는 방향으로 학습을 수행
  - 학습이 어느 정도 되고 나면 Attention Layer를 태어내 시각화 수행



## 4. 최종 모델 Deep Dive

### 4.3 Attention Mechanism 기반 Feature Importance 측정

- Feature Engineering
  - 목적
    - Noise 필드를 추가하여 기존 Feature 대비 활성화 여부를 조금 더 부각되어 보이게 하기 위함
    - 일반 원본 Feature 13개와 Noise 추가 Feature 26개를 비교하기 위함
- Input 종류



원본 13개 Feature

Noise Feature 13개



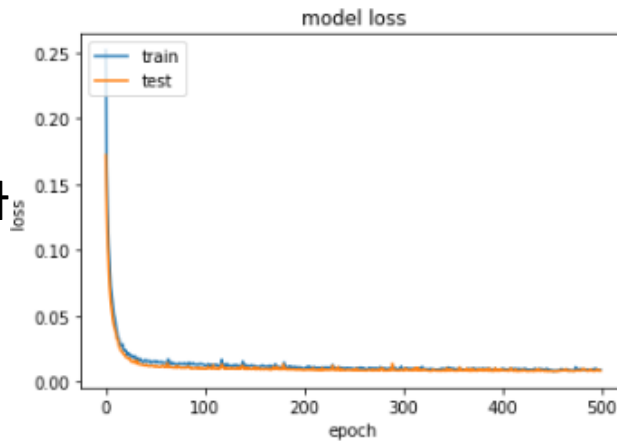
원본 13개 Feature

## 4. 최종 모델 Deep Dive

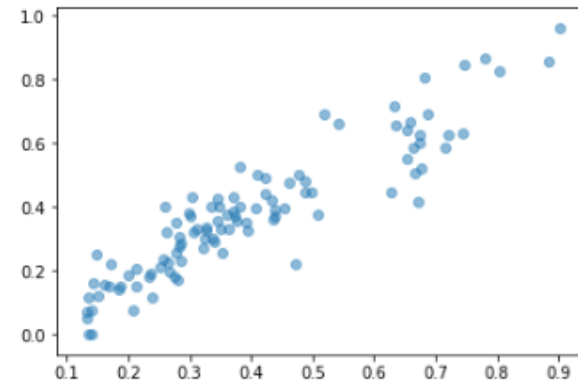
### 4.3 Attention Mechanism 기반 Feature Importance 측정

- 학습 결과

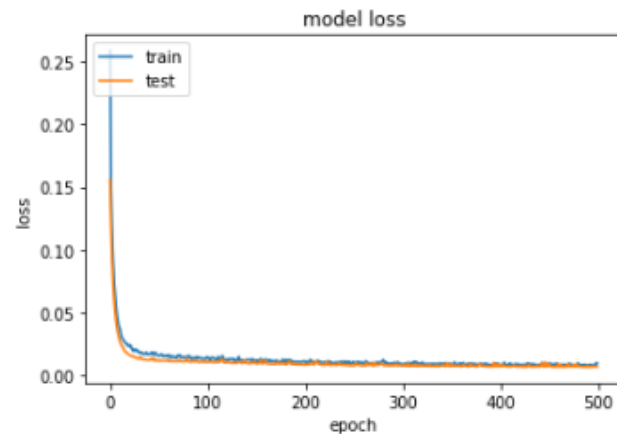
Noise 추가  
Feature



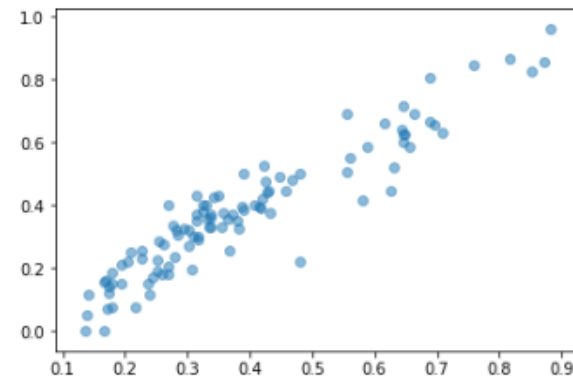
[INFO] MSE : 0.006674



원본 13개  
Feature



[INFO] MSE : 0.00602

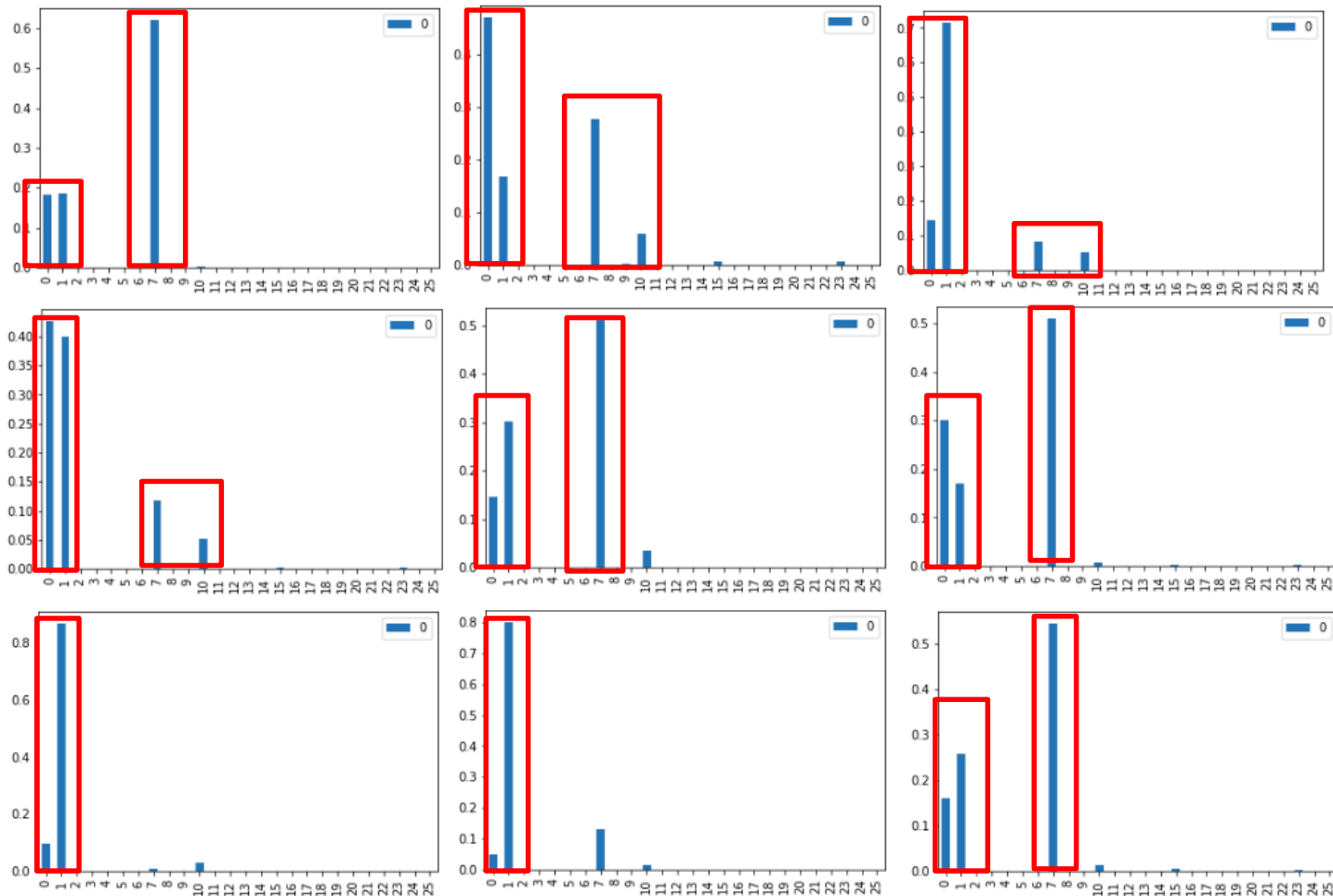


- 전반적으로 최종 모델의 성능보다 떨어지지만 어느 정도 예측을 잘 하는 것으로 보여짐

## 4. 최종 모델 Deep Dive

### 4.3 Attention Mechanism 기반 Feature Importance 측정

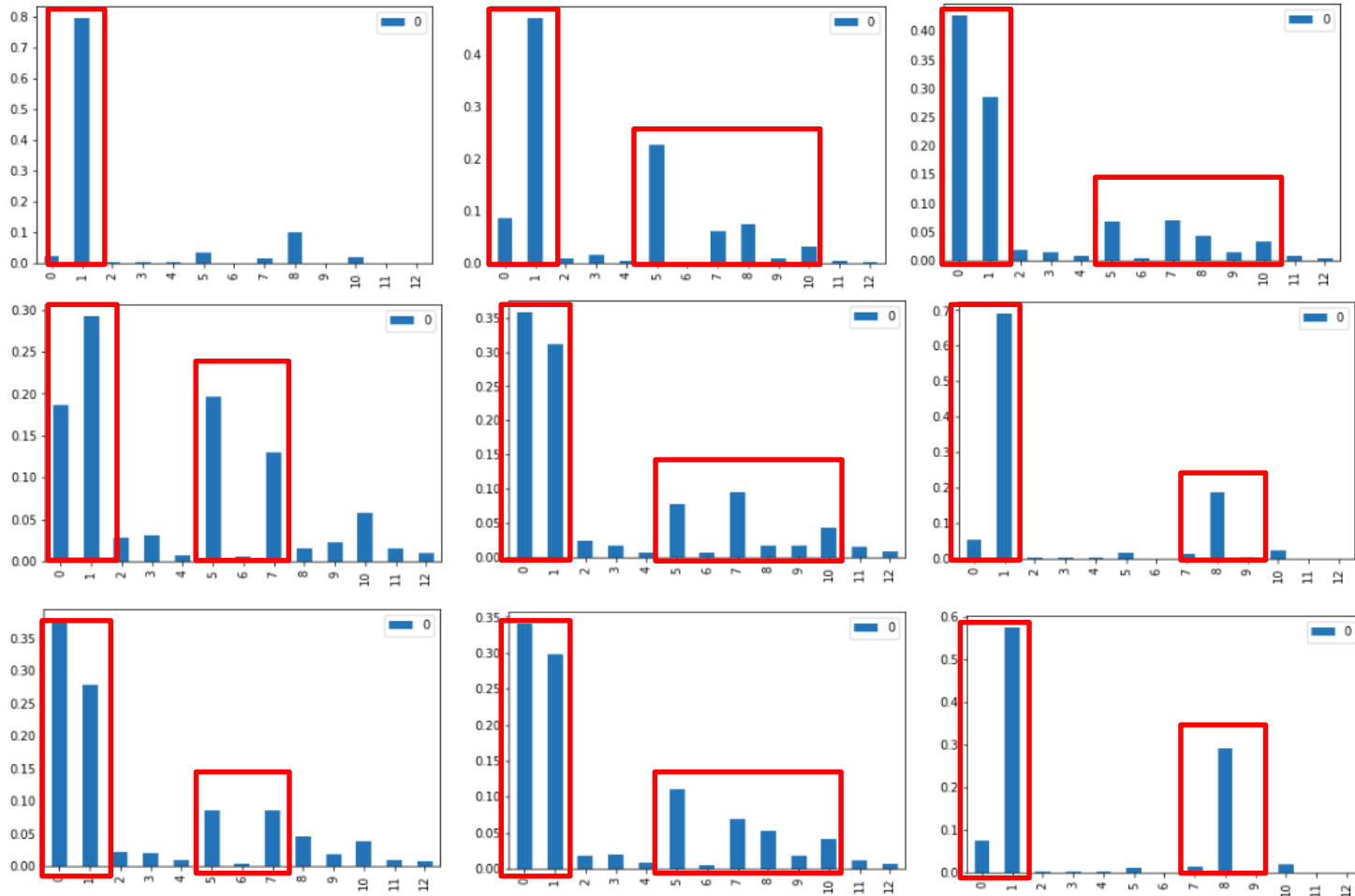
- 활성화 층 가시화 ( Noise 추가 Feature )



# 4. 최종 모델 Deep Dive

## 4.3 Attention Mechanism 기반 Feature Importance 측정

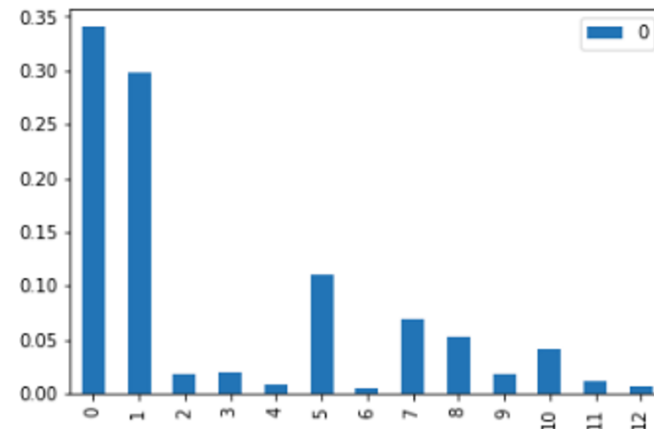
- 활성화 층 가시화 ( 원본 Feature 13개 )



## 4. 최종 모델 Deep Dive

### 4.3 Attention Mechanism 기반 Feature Importance 측정

- 활성화 층 분석결과
  - Input 종류에 관계 없이 활성화된 Feature는 한정적
  - Test 셋의 대부분의 Attention Layer는 아래 Feature가 활성화 되어 있음
    - RM
    - LSTAT
    - NOX
    - DIS
    - B
    - CHAS

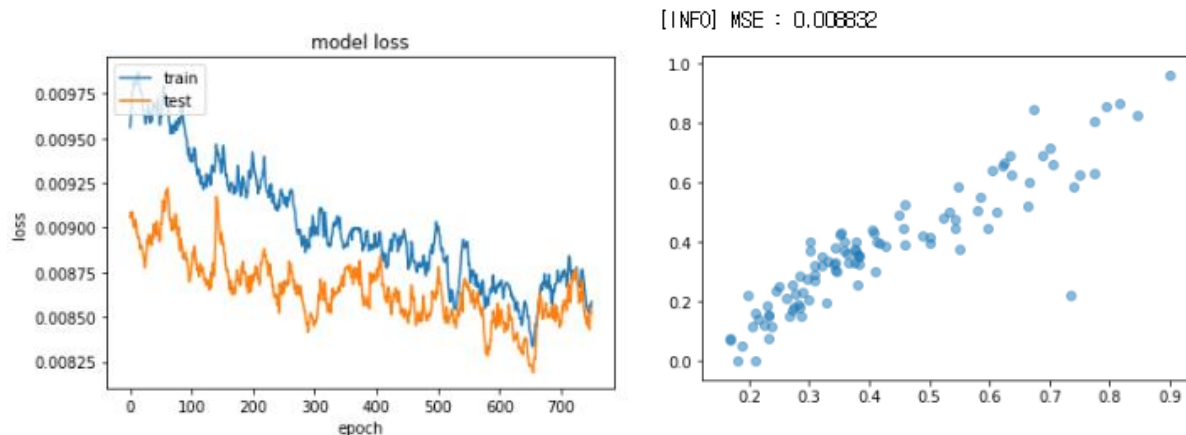


0	1	2	3	4	5	6	7	8	9	10	11	12
RM	LSTAT	PTRATIO	CRIM	INDUS	NOX	AGE	DIS	B	ZN	CHAS	RAD	TAX
0.354666	0.767660	0.638298	0.003183	0.371334	0.213992	0.071061	0.223508	0.879041	0.00	0	0.130435	0.171756
0.613336	0.214404	0.755319	0.001351	0.171188	0.139918	0.417096	0.623021	0.996672	0.25	0	0.304348	0.185115
0.589002	0.117550	0.468085	0.000098	0.056818	0.102881	0.276004	0.656039	0.984972	0.80	0	0.130435	0.177481
0.698410	0.175773	0.042553	0.006112	0.128666	0.539095	0.913491	0.072793	0.977281	0.20	0	0.173913	0.146947
0.818164	0.075055	0.553191	0.000559	0.073314	0.211934	0.522142	0.188198	0.989233	0.00	0	0.086957	0.011450

## 4. 최종 모델 Deep Dive

### 4.3 Attention Mechanism 기반 Feature Importance 측정

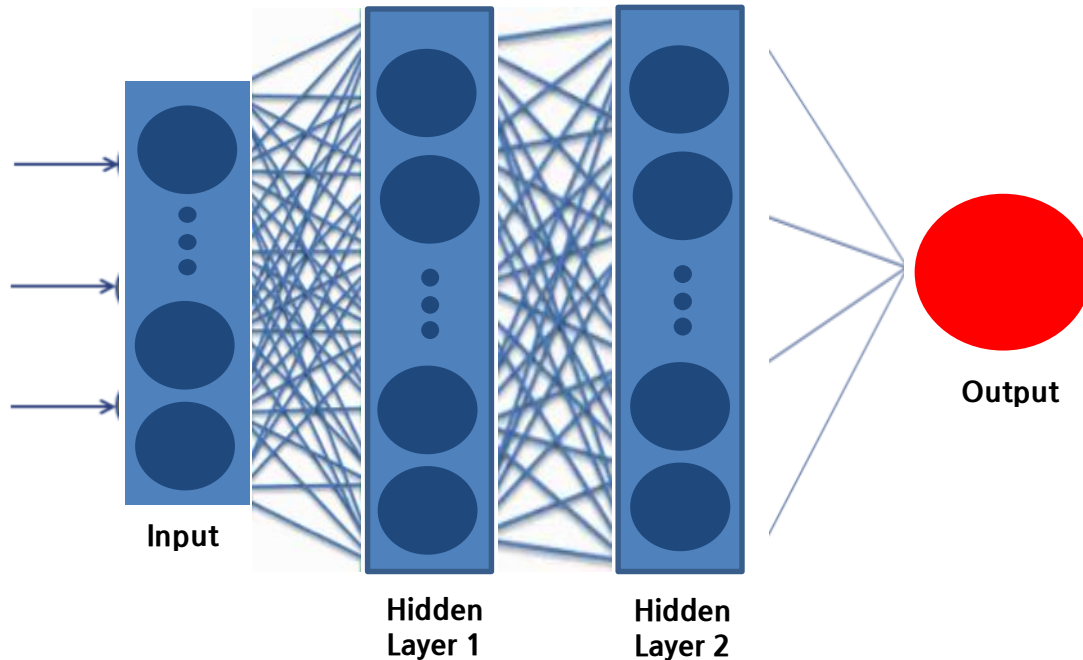
- Feature Selection
  - 목적
    - 집값을 결정하는 주요한 요소들만 Input을 구성하여 학습하기 위함
    - Attention Layer의 활성화 분석을 기반으로 Feature 재선정
- 학습 결과



- 새로 선택된 Feature로 학습 시 최종 모델보다는 좋은 성능을 보이지 못함
- 새로 선택된 Feature가 최종 모델에서 선택된 Feature 대부분에 포함되고, 포함되지 않은 추가적인 Feature와 결합되어 더 유의미한 결과를 내고 있는 것으로 판단됨

# 5. 결론

## 5.1 최종 모델



- 8 Input ( RM, LSTAT, PTRATIO, CRIM, INDUS, NOX, AGE, DIS)
- Min-Max Scaling (Normalization)
- Selective distribution transformation
- 2Hidden Layer ( 8 x64 x 64 x 1)
- Hyper Parameter
  - Mini Batch Size - 32, Learning rate - 0.001, Epoch - 500
- Drop Out (0.2%) , L2 Regularization, Xavier Initialize



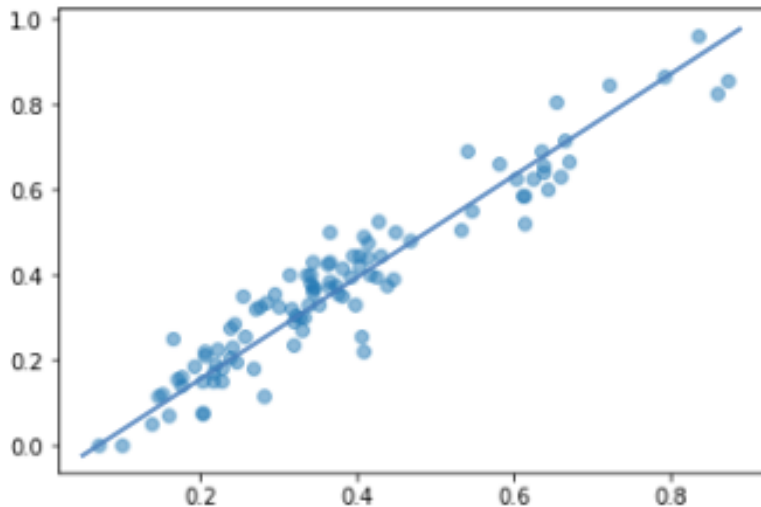
# 5. 결론

## 5.2 최종 모델 평가

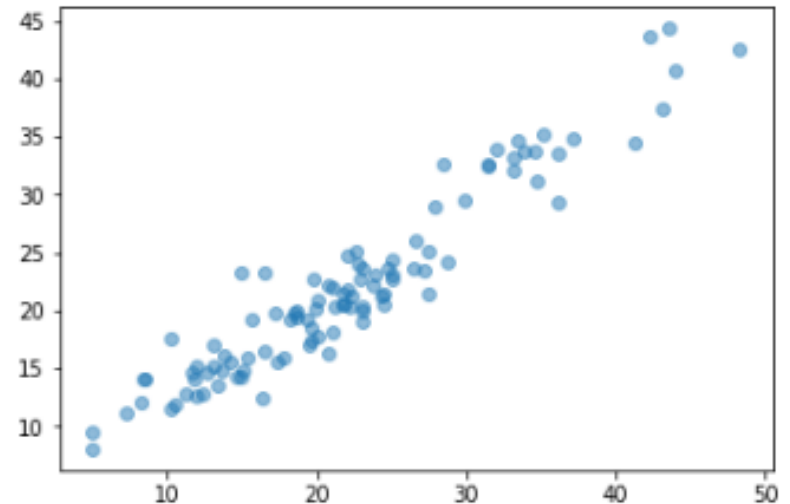
- 테스트 셋 예측 결과
  - 사전에 저장한 모델을 Reload하여 테스트 셋 대상으로 예측 수행
  - 예측 집값과 실제 집값이 Normalize되어 있기 때문에 이를 원 데이터로 복구

[INFO] MSE : 0.004276

Normalize 데이터



실제 집값 복구 데이터



# 5. 결론

## 5.2 최종 모델 평가

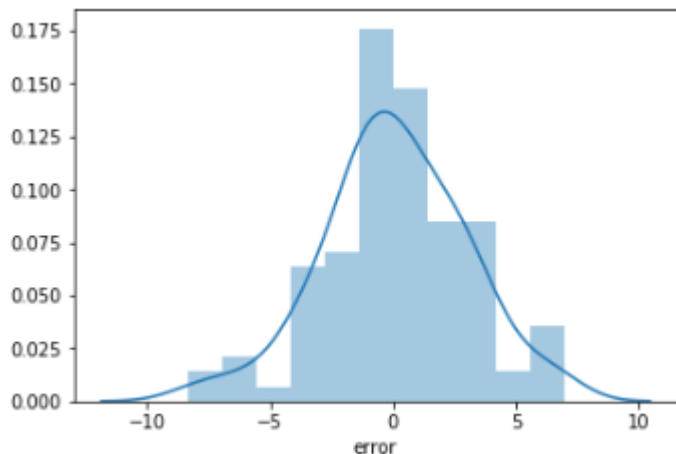
- 테스트 셋 예측 결과

### 최종 MSE

```
print(mean_squared_error(result.real_MEDV, result.pred_real_MEDV))
```

8.65816657411289

### error 분포



### 결과 샘플

실제  
집값      예측  
집값

	key	MEDV_x	pred_MEDV	real_MEDV	pred_real_MEDV	error
0	139	0.284444	0.243497	17.8	15.957350	1.842650
1	327	0.382222	0.340672	22.2	20.330231	1.869769
2	246	0.428889	0.361512	24.3	21.268055	3.031945
3	477	0.155556	0.168739	12.0	12.593248	-0.593248
4	426	0.115556	0.280725	10.2	17.632633	-7.432633
5	421	0.204444	0.235801	14.2	15.611064	-1.411064
6	157	0.806667	0.653305	41.3	34.398705	6.901295
7	100	0.500000	0.364697	27.5	21.411346	6.088654
8	208	0.431111	0.343144	24.4	20.441460	3.958540
9	132	0.400000	0.312613	23.0	19.067564	3.932436
10	169	0.384444	0.363075	22.3	21.338358	0.961642
11	155	0.235556	0.317556	15.6	19.290012	-3.690012
12	413	0.251111	0.164048	16.3	12.382182	3.917818
13	229	0.588889	0.613324	31.5	32.599556	-1.099556
14	81	0.420000	0.401078	23.9	23.048523	0.851477
15	85	0.480000	0.467079	26.6	26.018576	0.581424
16	494	0.433333	0.363264	24.5	21.346863	3.153137
17	504	0.377778	0.436967	22.0	24.663525	-2.663525
18	375	0.222222	0.407962	15.0	23.358278	-8.358278
19	149	0.231111	0.240857	15.4	15.838547	-0.438547
20	425	0.073333	0.158453	8.3	12.130367	-3.830367
21	2	0.660000	0.579785	34.7	31.090309	3.609691
22	334	0.348889	0.379632	20.7	22.083431	-1.383431
23	67	0.377778	0.373407	22.0	21.803326	0.196674
24	454	0.220000	0.204845	14.9	14.218007	0.681993
25	422	0.351111	0.252983	20.8	16.384253	4.415747
26	383	0.162222	0.173759	12.3	12.819166	-0.519166
27	483	0.373333	0.343118	21.8	20.440331	1.359669
28	323	0.300000	0.327383	18.5	19.732224	-1.232224
29	172	0.402222	0.339915	23.1	20.296165	2.803835

# 5. 결론

## 5.2 최종 모델 평가

- Kaggle Best Voting Project와의 비교
  - 동일 개수의 예측 데이터를 사용하여 평가 수행

22 House sales price using Regression 2y ago

8 The Boston Housing Dataset 1y ago

3 MORE ABOUT LINEAR REGRESSION 2mo ago knowledge, data visualization, linear regression, regression, future prediction

2 Boston house price prediction 1mo ago beginner, eda, regression, starter code

2 boston house price with keras 10mo ago

Make predictions on validation dataset

Our Model is ready for prediction with our Validation set

```
# prepare the model
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
model = GradientBoostingRegressor(random_state=seed, n_estimators=400)
model.fit(rescaledX, Y_train)
# transform the validation dataset
rescaledValidationX = scaler.transform(X_validation)
predictions = model.predict(rescaledValidationX)
print(mean_squared_error(Y_validation, predictions))
```

26.3285956916

26.3285956916 VS 8.6581665741

실험한 최종 DNN 모델이 더 좋은 성능을 보임

## 5. 결론

### 5.4 아쉬운 점

---

- Data Augmentation
  - 본래 계획은 데이터를 증가 시켜 조금 더 일반화된 모델을 만들어 보고자 했음
- DNN 모델에 편중
  - Gradient Boost 모델을 깊게 파보지 않고 DNN 모델에 선부르게 Deep Dive 한 것이 아닌가라는 의심
  - 오히려 Gradient Boost 모델이 더 좋은 결과를 낼 수도 있었을 것
- Chapter 4.3 Feature추가 및 변경에 따른 모델 튜닝 부분을 깊게 파보지 않은 것
  - 해당 부분에서 좋은 결과가 나올 수도 있었을 것
- Attention Layer에서 추출된 Feature에 대해 깊게 분석해 보지 않은 점
  - 8개 Feature보다 Attention Layer에서 활성화된 6개가 오히려 Fitting된 결과가 나올 수도 있었을 것

## 6. Reference

- 프랑소와 솔레, 케라스 창시자에게 배우는 딥러닝, 길벗 ( 2018 )
- <https://github.com/philipperemy/keras-attention-mechanism>
- <https://www.kaggle.com/vikrishnan/house-sales-price-using-regression>

감사합니다