8th Meet

□ 오늘 공부할 것은

05

회귀

Now, It's your turn !!!

40min ~ 50min

Chapter 05

회귀 (Regression)

[05-06]

규제 선형 모델

- 릿지, 라쏘, 엘라스틱넷

□ 규제 선형 모델의 개요

최적 모델을 위한 Cost 함수 구성 요소

학습데이터 잔차 오류 최소화	+	회귀계수 크기 제어

학습 데이터 최적화 → 회귀계수 크기 증가 → overfitting

비용 함수의 목표 → $Min(RSS(W) + \underline{\alpha * \|W\|_2^2})$

회귀계수 크기를 조절하기 위한 부분

→ Regularization (규제)

□ 릿지 (Ridge, L2-Regularization)

$$y = \omega_1 * x_1 + \omega_2 * x_2 + \omega_3 * x_3 + \ldots$$

$$\omega_1^2 + \omega_2^2 + \omega_3^2 + \ldots \leq \boldsymbol{R}$$

https://github.com/whatwant-school/python-ml/blob/main/07-week/07-week_07-Regularization.ipynb

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
import numpy as np

ridge = Ridge(alpha = 10)
neg_mse_scores = cross_val_score(ridge, data_X, target_y, scoring="neg_mean_squared_error", cv=5)

rmse_scores  = np.sqrt(-1 * neg_mse_scores)
avg_rmse = np.mean(rmse_scores)

print(' 5 folds 의 개별 Negative MSE scores: ', np.round(neg_mse_scores, 3))
print(' 5 folds 의 개별 RMSE scores : ', np.round(rmse_scores,3))
print(' 5 folds 의 평균 RMSE : {0:.3f} '.format(avg_rmse))
```

```
 5 folds 의 개별 Negative MSE scores:  [-16.419 -26.269 -34.91  -92.726 -58.468]
 5 folds 의 개별 RMSE scores :  [4.052 5.125 5.909 9.629 7.646]
 5 folds 의 평균 RMSE : 6.472
```

□ 라쏘 (Lasso, L1-Regularization)

$$y = \omega_1 * x_1 + \omega_2 * x_2 + \omega_3 * x_3 + \dots$$

$$|\omega_1| + |\omega_2| + |\omega_3| + \dots \le \boldsymbol{R}$$

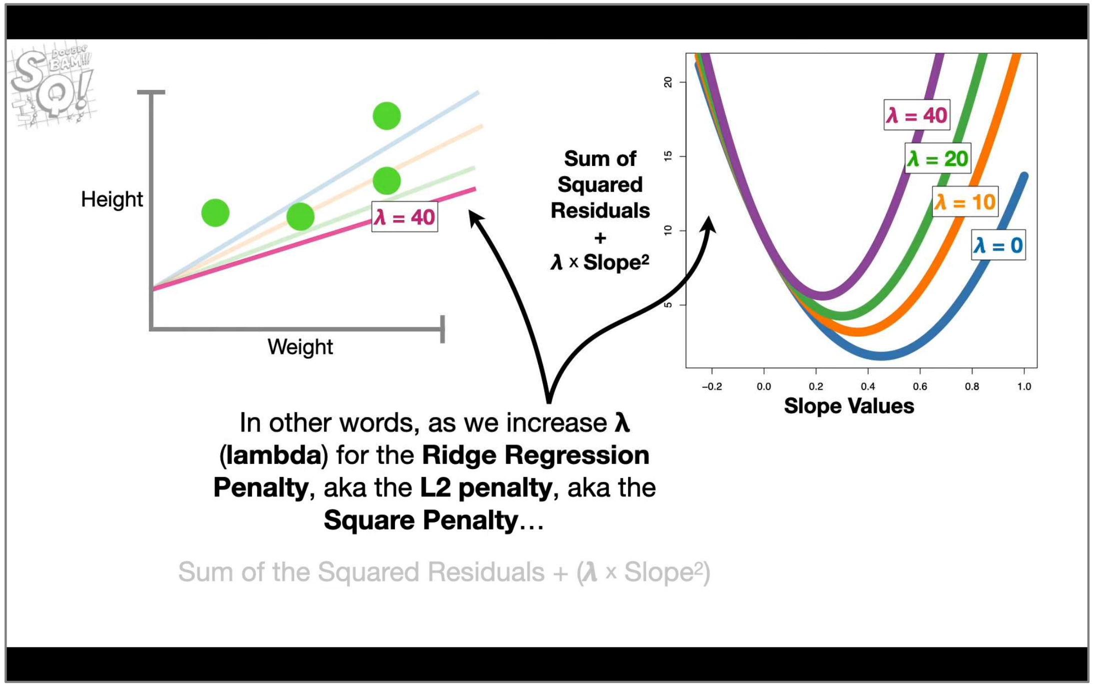
https://github.com/whatwant-school/python-ml/blob/main/07-week/07-week_07-Regularization.ipynb

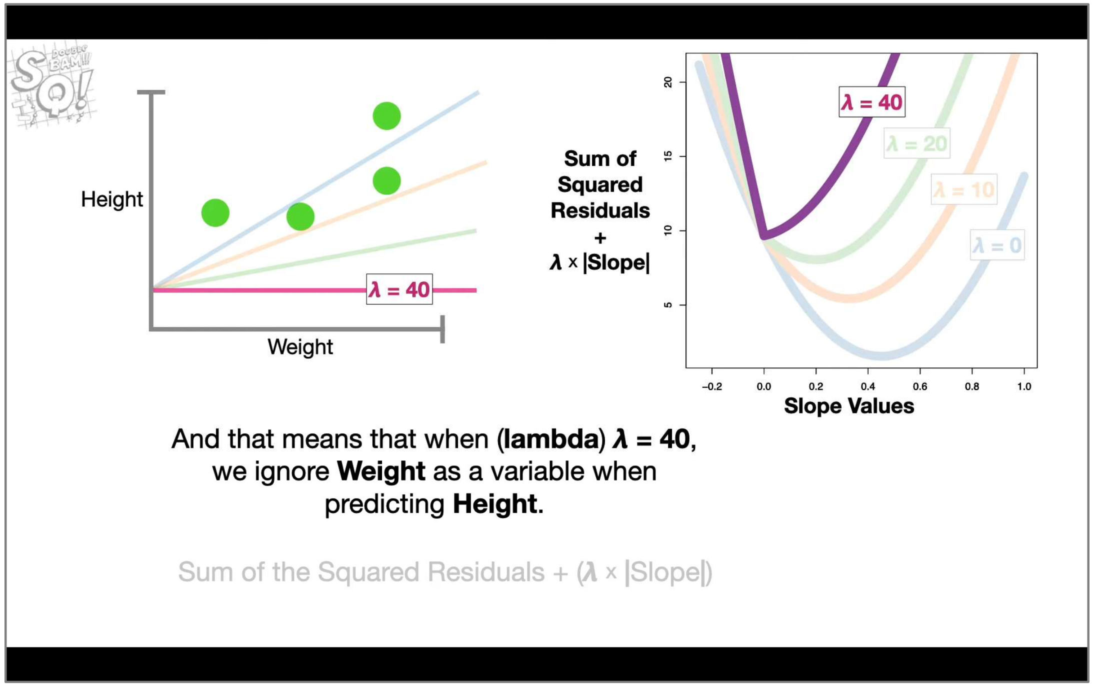
```
lasso_alphas = [ 0.07, 0.1, 0.5, 1, 3]

coeff_lasso_df = get_linear_reg_eval('Lasso', params=lasso_alphas, data_X_n=data_X, target_y_n=target_y)
```

```
#######  Lasso #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 6.676
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 6.694
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 6.792
alpha 1일 때 5 폴드 세트의 평균 RMSE: 6.874
alpha 3일 때 5 폴드 세트의 평균 RMSE: 7.289
```

☐ Ridge & Lasso

https://www.youtube.com/watch?v=Xm2C_gTAI8c





□ 엘라스틱넷 (Elastic Net, L1+L2-Regularization)

$$Min(RSS(W) + \lambda_1 * \|W\|_2^2 + \lambda_2 * \|W\|_1)$$

$$y = \omega_1 * x_1 + \omega_2 * x_2 + \omega_3 * x_3 + \ldots$$

$$(\omega_1^2 + \omega_2^2 + \omega_3^2 + \ldots) + (|\omega_1| + |\omega_2| + |\omega_3| + \ldots) \leq \boldsymbol{R}$$

https://github.com/whatwant-school/python-ml/blob/main/07-week/07-week_07-Regularization.ipynb

```
elastic_alphas = [ 0.07, 0.1, 0.5, 1, 3]

coeff_elastic_df = get_linear_reg_eval('ElasticNet', params=elastic_alphas, data_X_n=data_X, target_y_n=target_y)
```

```
#######  ElasticNet #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 6.567
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 6.550
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 6.451
alpha 1일 때 5 폴드 세트의 평균 RMSE: 6.552
alpha 3일 때 5 폴드 세트의 평균 RMSE: 7.122
```

□ 선형 회귀 모델을 위한 데이터 변환

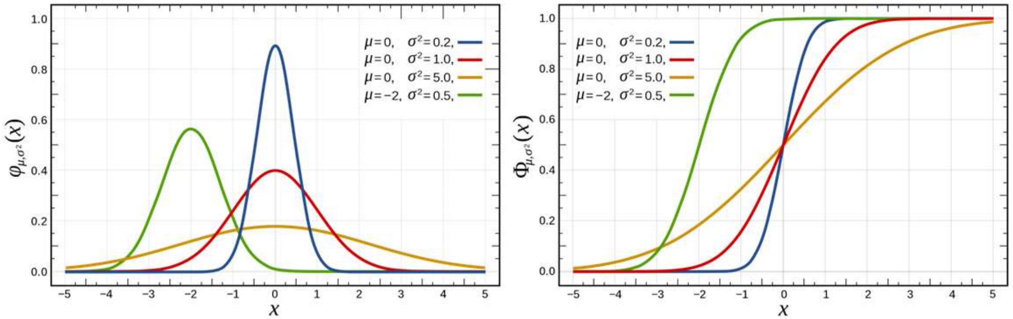
- 선형 회귀 모델은 features, label 모두 **정규 분포** 선호

Case 1
- StandardScaler : 평균 = 0, 분산 = 1
- MinMaxScaler : 최솟값 = 0, 최댓값 = 1

Case 2
- Scaling/Normalization 한 데이터에 다시 다항 특성을 적용하여 변환

Case 3
- Log Transformation (로그 변환)

[확률 밀도 함수] [누적 분포 함수]



※ 출처 : https://ko.wikipedia.org/wiki/정규_분포

https://github.com/whatwant-school/python-ml/blob/main/07-week/07-week_08-Transform.ipynb

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures
import numpy as np

def get_scaled_data(method='None', p_degree=None, input_data=None):
    if method == 'Standard':
        scaled_data = StandardScaler().fit_transform(input_data)

    elif method == 'MinMax':
        scaled_data = MinMaxScaler().fit_transform(input_data)

    elif method == 'Log':
        scaled_data = np.log1p(input_data)
```

[05-07]

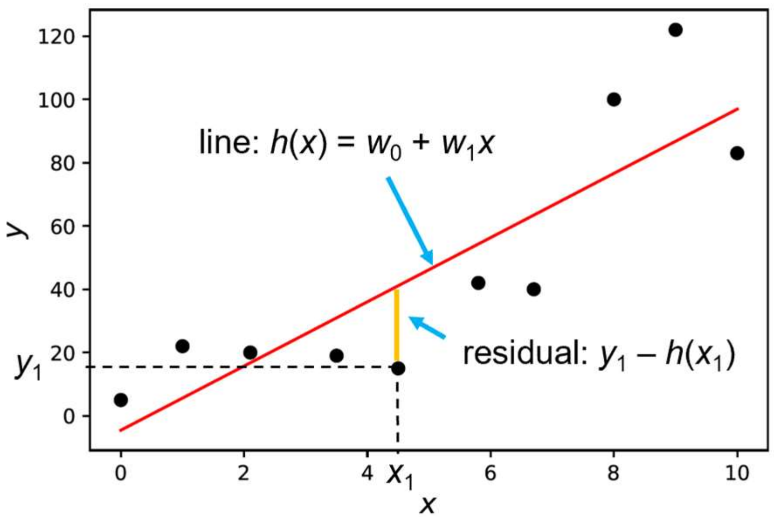
로지스틱 회귀

(Logistic Regression)

☐ Linear Regression vs. Logistic Regression

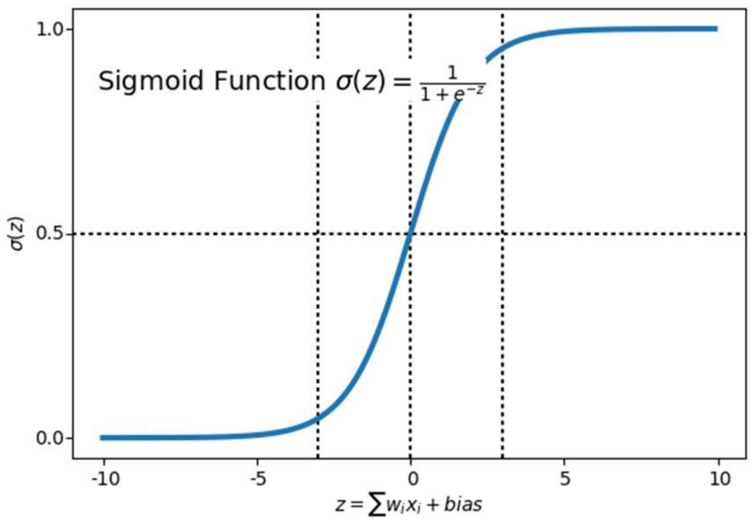
Linear Regression

$$\hat{y} = \omega_0 + \omega_1 * x_1 + \omega_2 * x_2 + \ldots$$



Logistic Regression

$$\hat{y} = sigmoid(w^T x + b)$$



☐ solver

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

solver : *{'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}, default='lbfgs'*

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

- For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones;
- For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss;
- 'liblinear' is limited to one-versus-rest schemes.
- 'newton-cholesky' is a good choice for `n_samples` >> `n_features`, especially with one-hot encoded categorical features with rare categories. Note that it is limited to binary classification and the one-versus-rest reduction for multiclass classification. Be aware that the memory usage of this solver has a quadratic dependency on `n_features` because it explicitly computes the Hessian matrix.

Warning: The choice of the algorithm depends on the penalty chosen. Supported penalties by solver:

- 'lbfgs' - ['l2', None]
- 'liblinear' - ['l1', 'l2']
- 'newton-cg' - ['l2', None]
- 'newton-cholesky' - ['l2', None]
- 'sag' - ['l2', None]
- 'saga' - ['elasticnet', 'l1', 'l2', None]

Note: 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from `sklearn.preprocessing`.

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

	Solvers					
Penalties	**'lbfgs'**	**'liblinear'**	**'newton-cg'**	**'newton-cholesky'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	yes	no	yes	no	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	no	yes
OVR + L1 penalty	no	yes	no	no	no	yes
Elastic-Net	no	no	no	no	no	yes
No penalty ('none')	yes	no	yes	yes	yes	yes
Behaviors						
Penalize the intercept (bad)	no	yes	no	no	no	no
Faster for large datasets	no	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	yes	no	no

☐ Logistic Regression

https://github.com/whatwant-school/python-ml/blob/main/08-week/08-week_01-LogisticRegression.ipynb

```
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression()
lr_clf
```

```
LogisticRegression

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
from sklearn.metrics import accuracy_score, roc_auc_score

lr_clf.fit(train_X, train_y)
predicts = lr_clf.predict(test_X)
probas = lr_clf.predict_proba(test_X)[:, 1]

print('accuracy: {0:.3f}, roc_auc:{1:.3f}'.format(accuracy_score(test_y, predicts),
                                                  roc_auc_score(test_y, probas)))
```

```
accuracy: 0.982, roc_auc:0.998
```

[05-08]
회귀 트리
(Regression Tree)

☐ Tree

split 0
split 1
split 2
split 3
Y
X
1
2
3
4
3.2
변환
X > 2
No
Yes
X > 2
X > 3.2
X > 4

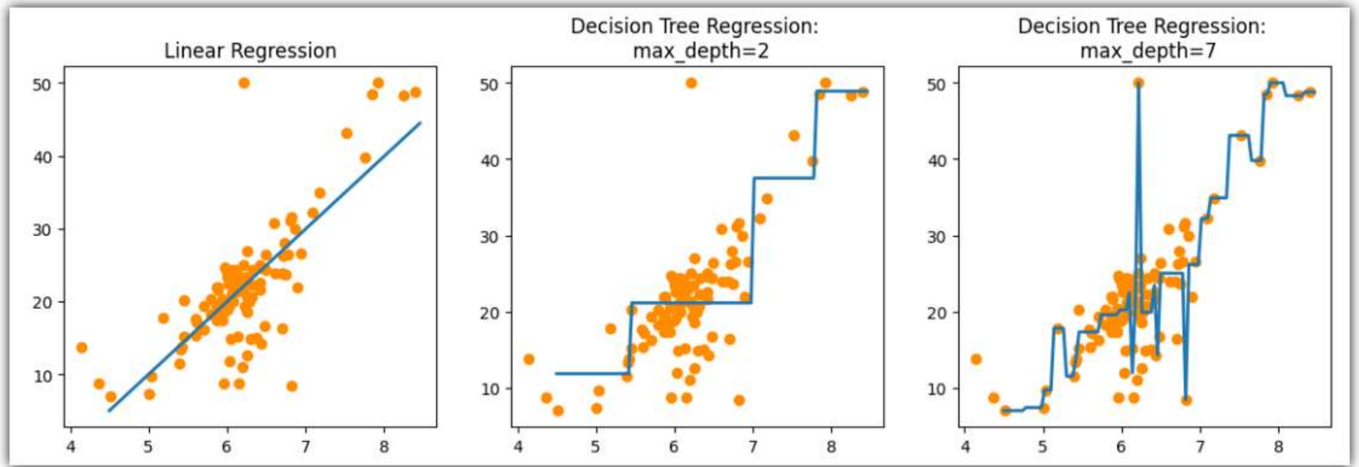
split 0
split 1
split 2
split 3
Y
X
1
2
3
4
3.2
평균 2.5
평균 3
평균 2.7
평균 3.5
평균 4.2
예측값 부여
X > 2
No
Yes
X > 2
X > 3.2
X > 4
2.5
3
2.7
3.5
4.2

☐ Table

Algorithm	Regression	Classification
Decision Tree	DecisionTreeRegressor	DecisionTreeClassifier
Gradient Boosting	GradientBoostingRegressor	GradientBoostingClassifier
XGBoost	XGBRegressor	XGBClassifier
LightGBM	LGBMRegressor	LGBMClassfier

☐ Code

https://github.com/whatwant-school/python-ml/blob/main/08-week/08-week_02-Tree.ipynb

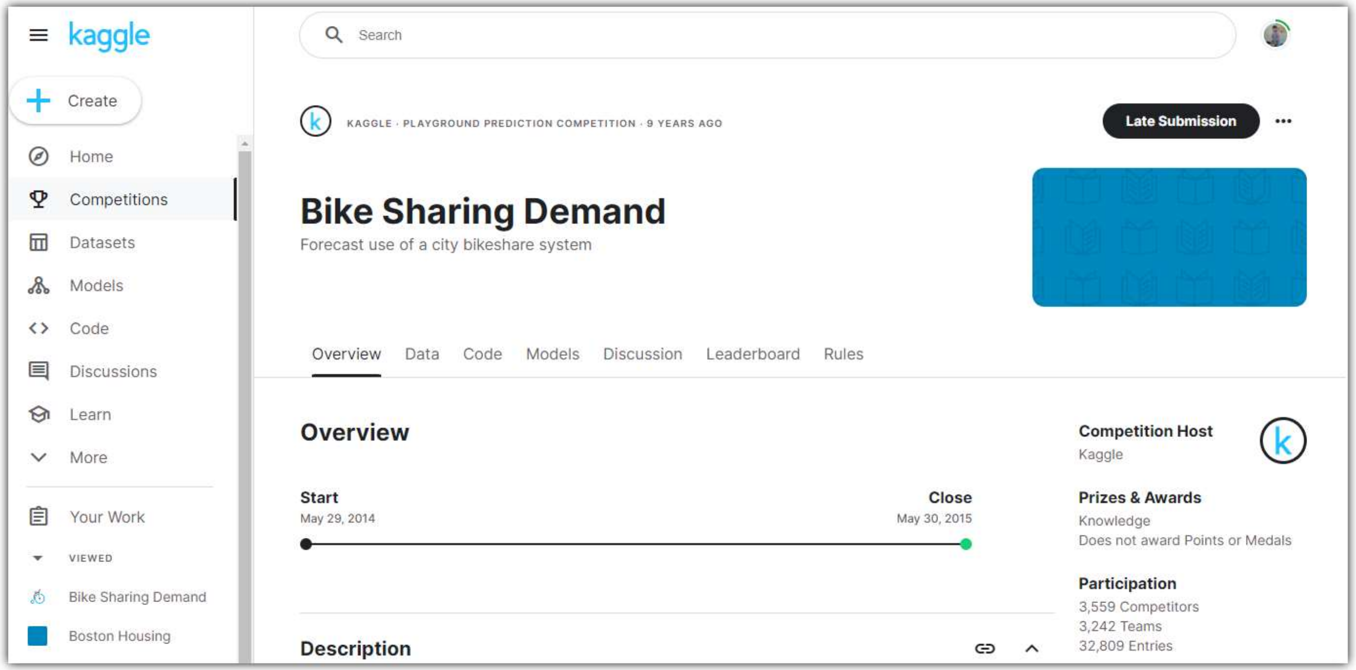


[05-09]
회귀 실습
- 자전거 대여 수요 예측

☐ Bike Sharing Demand

- 워싱턴 DC의 Capital Bikeshare 프로그램에서 과거 사용 패턴과 날씨 데이터를 결합하여 자전거 대여 수요를 예측하기
- Capital Bikeshare 프로그램은 회원 가입, 대여, 자전거 반납 과정이 자동화되는 자전거 대여 수단
- 데이터는 여행 기간, 출발 위치, 도착 위치 및 경과 시간이 명시적으로 기록 → 도시의 이동성을 연구하는 데 사용 가능

https://www.kaggle.com/c/bike-sharing-demand



☐ Bike Sharing Demand

https://github.com/whatwant-school/python-ml/blob/main/08-week/08-week_03-bike.ipynb

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

rf_reg = RandomForestRegressor(n_estimators=500)
gbm_reg = GradientBoostingRegressor(n_estimators=500)
xgb_reg = XGBRegressor(n_estimators=500)
lgbm_reg = LGBMRegressor(n_estimators=500)

for model in [rf_reg, gbm_reg, xgb_reg, lgbm_reg]:
    get_model_predict(model, train_X.values, test_X.values, train_y.values, test_y.values, is_expm1=True)
```

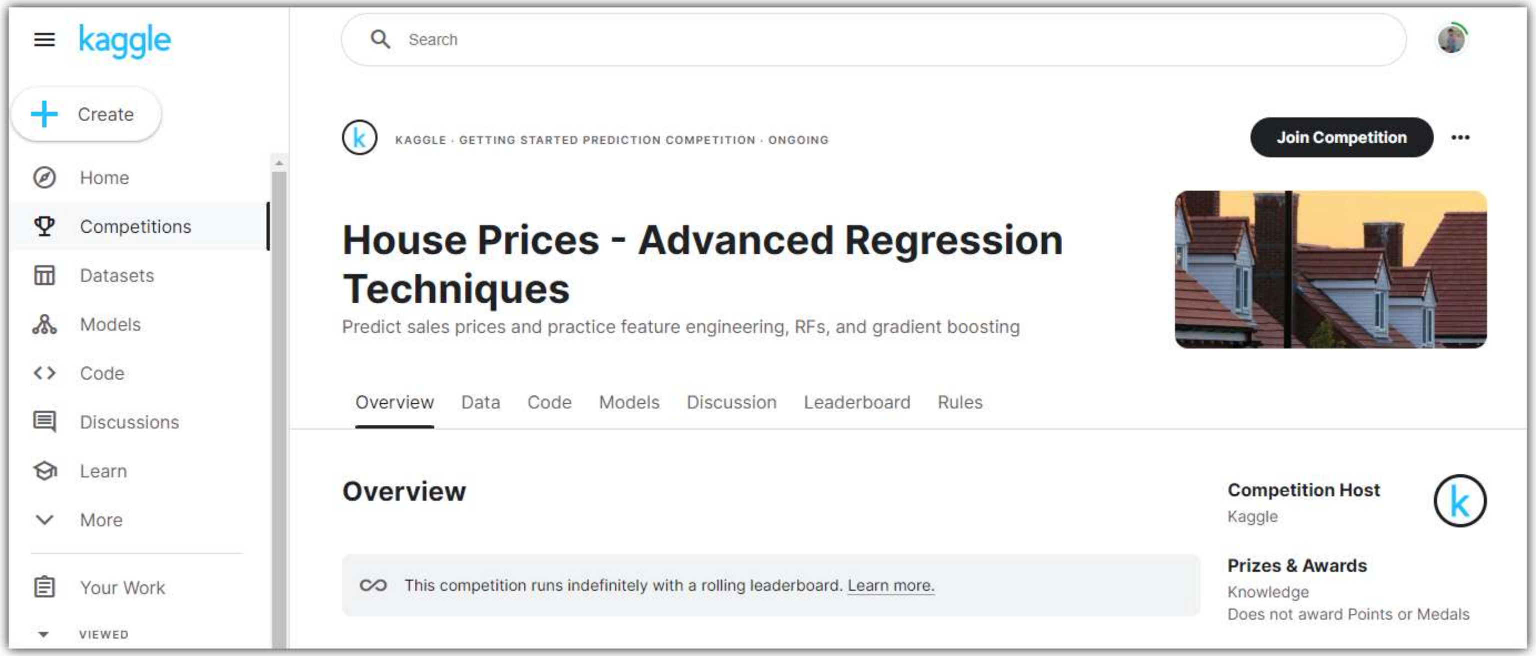
```
### RandomForestRegressor ###
RMSLE: 0.349, RMSE: 48.095, MAE: 30.091
### GradientBoostingRegressor ###
RMSLE: 0.331, RMSE: 51.697, MAE: 32.198
### XGBRegressor ###
RMSLE: 0.344, RMSE: 52.341, MAE: 32.070
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000252 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 347
[LightGBM] [Info] Number of data points in the train set: 7620, number of used features: 72
[LightGBM] [Info] Start training from score 4.585795
### LGBMRegressor ###
RMSLE: 0.314, RMSE: 44.509, MAE: 27.586
```

[05-10]
회귀 실습
- 캐글 주택 가격: 고급 회귀 기법

□ House Prices - Advanced Regression Techniques

– 아이오와(Iowa) 주 에임스(Ames)에 있는 주거용 주택을 설명하는 79개의 변수를 사용하여 주택의 가격을 예측

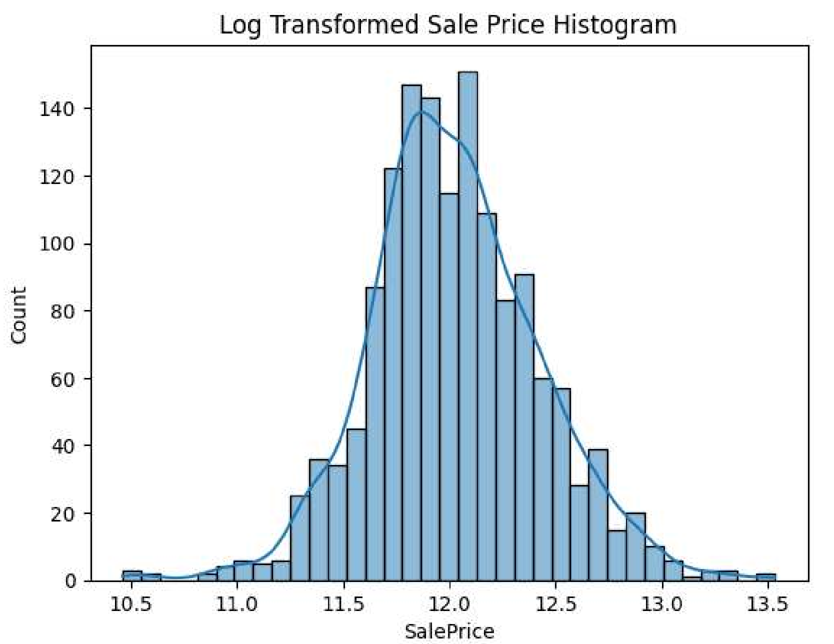
https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques



☐ House Prices - Advanced Regression Techniques

https://github.com/whatwant-school/python-ml/blob/main/08-week/08-week_04-house.ipynb

```
import numpy as np

plt.title('Log Transformed Sale Price Histogram')
log_SalePrice = np.log1p(df['SalePrice'])
sns.histplot(log_SalePrice, kde=True)
plt.show()
```



You've really worked hard today

Next Week ~ ?

Who ~ ?

See you Next Weekend ~ ?