Capstone Project Final Report

20186029 이승민

융합경영대학원 BA과정 (Part)

1. Background

S-OIL 주요 분별증류타워 전단에 위치한 히터(Heater)는 Tower에 유입되는 물질 (Crude, Petrochemical 등 Hydrocarbons)을 미리 가열하여 Tower로 공급함으로써, Tower의 열 효율을 보존/향상시키는 역할을 한다. 히터의 연료는 Methane/ Ethane/ Propane/ Butane 등의 Gas 성분으로 이뤄져 있으며, 공장 내 Cracking 공정 및 Stripping 공정 등에서 발생되는 Gas들이 Fuel Gas Header Line로 모아져서 각 공정의 히터에 연료로 쓰이게 된다.

문제는 Gas가 생산되는 여러 공정의 운전 조건에 따라 Gas의 조성과 발열량이 실시간으로 달라지기 때문에 히터의 연료 사용량을 적절히 조절하기가 쉽지 않다. 히터의 주요 운전 Target은 안정적인 히터 출구 온도 유지, 완전 연소를 위한 히터 내 Excess O2 적정량 유지 등이 있는데, Fuel Gas의 열량이 변동될 때마다 사용량을 적절히 제어하지 않으면 히터 출구 온도 또는 Excess O2 양이 급격히 변동될 수 있다.

본 프로젝트는 공정의 여러 운전 조건과 Fuel Gas 열량 간 상관관계를 파악하고 Data Analysis를 통해 최종적으로 Fuel Gas의 열량을 실시간 예측하는 것에 목적이 있다. 예측 성능이 양호할 경우 현재 본인이 담당하고 있는 공정자동화 제어시스템(Advanced Process Control System)에 예측 Model을 반영하여 실제 공정 운전에 반영하고자 한다.

- 대상 공정 : S-OIL Para-Xylene (PX) 생산 공정 Fractionator 전단 Heater
  - 하루 평균 약 60,000배럴(약 953만 리터)의 Xylene 성분을 처리하는 공정이며, Heater에 소모되는 연료(Fuel Gas)의 가치는 하루 평균 약 8천만원 수준임. (연간 280억원)
  - 타 Heater 대비 설비의 사이즈가 크고, 연료 사용량도 많아 Fuel Gas 열량 변화에 민감하며 실제 운전에 어려움이 많기 때문에 위 Heater를 분석 Target으로 정하였음.

2. 데이터 설명 (EDA 포함)

분석을 위한 Raw Data 정보는 아래와 같음.

- Data 수집 기간 : 2019-01-01 ~ 2020-03-27 (Daily basis, 451 days)
- Rows/Columns : 451 rows, 26 columns (Input 변수 25개, Target 변수 1개)
- Data Type : Input/Target 변수 모두 연속형 변수
- Input 변수 설명 (파생변수 포함)

1. FG_Press_A : Heater 내 A Box로 주입되는 Fuel Gas의 압력 (Unit : Kg/cmg)
2. FG_Press_B : Heater 내 B Box로 주입되는 Fuel Gas의 압력 (Unit : Kg/cmg)
3. FG_Press_Sum : Heater 내 A/B Box로 주입되는 Fuel Gas 압력의 합 (Unit : Kg/cmg) – 파생변수
4. FG_Flow_A : Heater 내 A Box로 주입되는 Fuel Gas의 물량 (Unit : Nm3/hr)
5. FG_Flow_B : Heater 내 B Box로 주입되는 Fuel Gas의 물량 (Unit : Nm3/hr)
6. FG_Flow_Sum : Heater 내 A/B Box로 주입되는 Fuel Gas 물량의 합 (Unit : Nm3/hr) – 파생변수
7. HTR_Flow : Heater의 Feed 물량 (Unit : Barrel/day)
8. HTR_Inlet_Temp_r1 : Heater Middle Section의 Inlet 온도 (Unit : Deg. C)
9. HTR_Outlet_Temp_r1_A : Heater 내 A Box의 Outlet 온도 (Unit : Deg. C)
10. HTR_Outlet_Temp_r1_B : Heater 내 B Box의 Outlet 온도 (Unit : Deg. C)
11. HTR_Delta_Temp_r1 : Heater Middle Section의 Outlet/Inlet 온도 차 (Unit : Deg. C) – 파생변수
12. HTR_Inlet_Temp : Heater Top Section의 Inlet 온도 (Unit : Deg. C)
13. HTR_Outlet_Temp : Heater Top Section의 Outlet 온도 (Unit : Deg. C)
14. HTR_Delta_Temp : Heater Top Section의 Outlet/Inlet 온도 차 (Unit : Deg. C) – 파생변수
15. HTR_O2_A : Heater 내 A Box의 Excess O2 함량 (Unit : vol%)
16. HTR_O2_B : Heater 내 B Box의 Excess O2 함량 (Unit : vol%)
17. HTR_O2_Sum : Heater 내 A/B Box의 Excess O2 함량의 합 (Unit : vol%) – 파생변수
18. FG_LHV : Heater Top Section 기준 FG 사용량 대비 온도 변화량 (Unit : None) – 파생변수
19. FG_LHV_r1 : Heater Middle Section 기준 FG 사용량 대비 온도 변화량 (Unit : None) – 파생변수
20. Offgas_S1 : Fuel Gas Header Line으로 유입되는 타 공정 Off-Gas 물량 1 (Unit : Nm3/hr)

21. Offgas_S2 : Fuel Gas Header Line으로 유입되는 타 공정 Off-Gas 물량 2 (Unit : Nm3/hr)
22. Offgas_S3 : Fuel Gas Header Line으로 유입되는 타 공정 Off-Gas 물량 3 (Unit : Nm3/hr)
23. Offgas_S4 : Fuel Gas Header Line으로 유입되는 타 공정 Off-Gas 물량 4 (Unit : Nm3/hr)
24. Offgas_S5 : Fuel Gas Header Line으로 유입되는 타 공정 Off-Gas 물량 5 (Unit : Nm3/hr)
25. Offgas_S6 : Fuel Gas Header Line으로 유입되는 타 공정 Off-Gas 물량 6 (Unit : Nm3/hr)

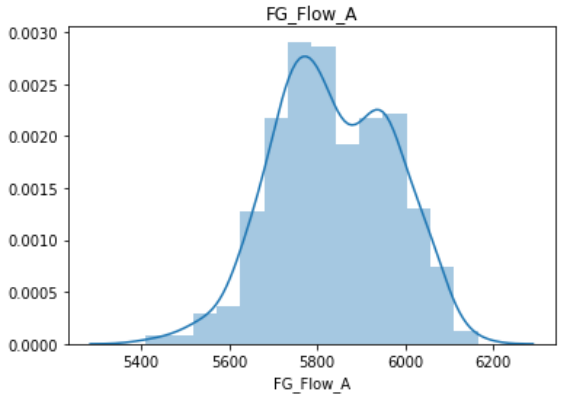
➢ Target 변수 설명

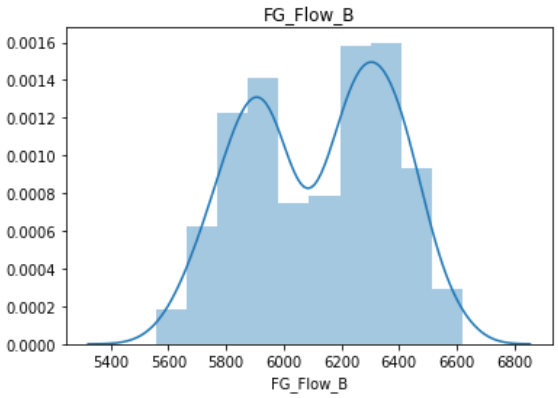
1. FG_LHV_Lab : Heater로 주입되는 Fuel Gas 열량 분석 결과 (Unit : BTU/Nm3)(Unit : BTU/Nm3)

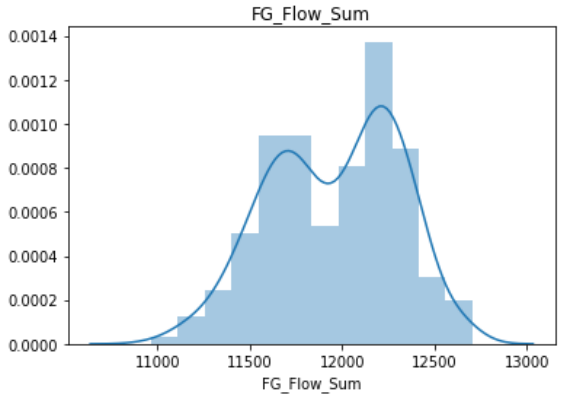
   * Fuel Gas 열량은 주 1회 실험실에서 조성 분석을 실시하며 조성에 따른 열량을 환산하였음.

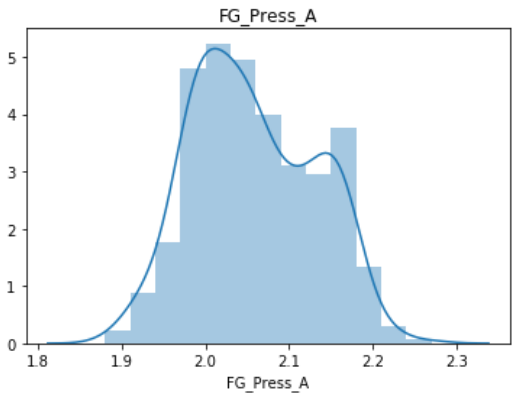
Exploratory Data Analysis

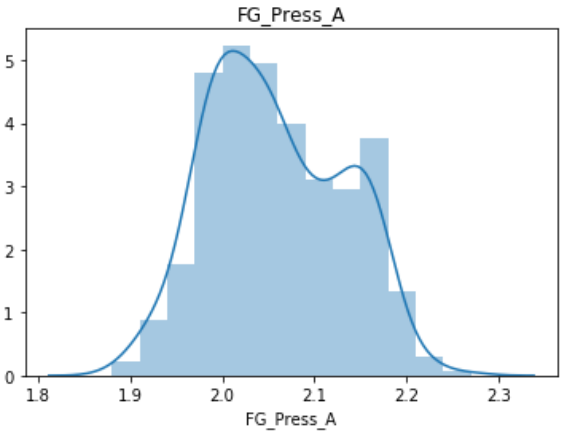
➢ Input Variable Distribution

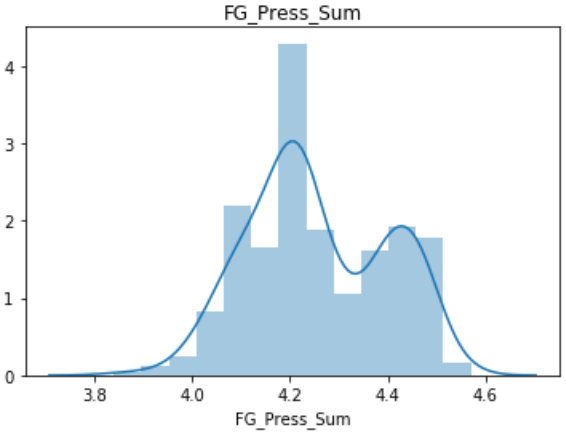


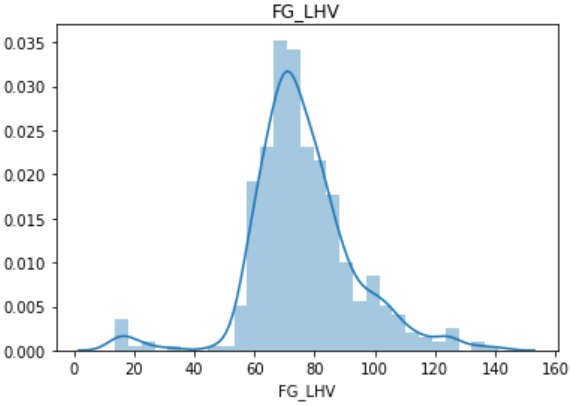


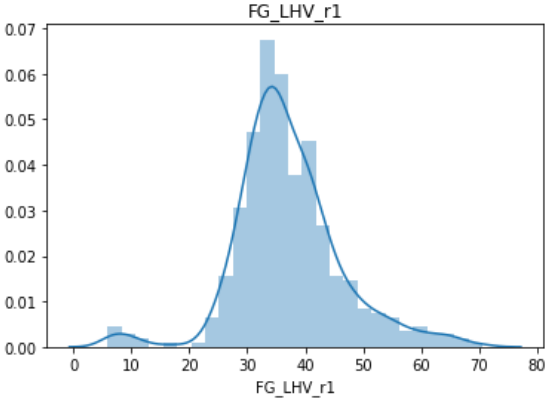


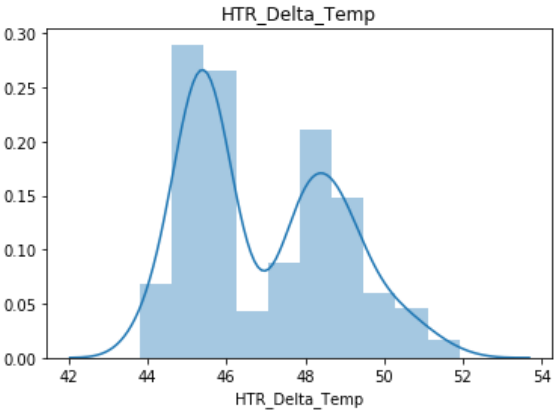


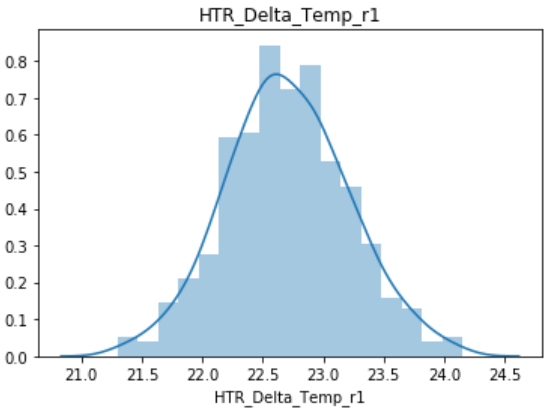

FG_Press_A
FG_Press_A

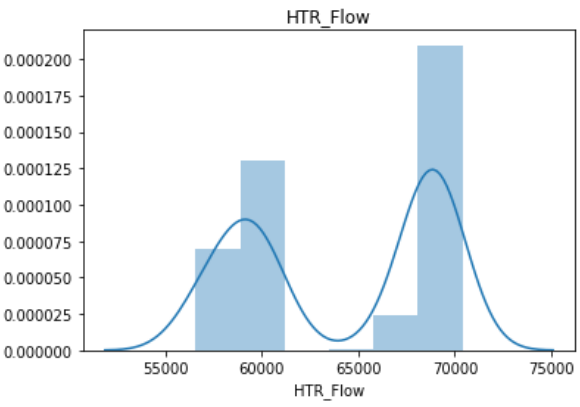

FG_Press_Sum
FG_Press_Sum

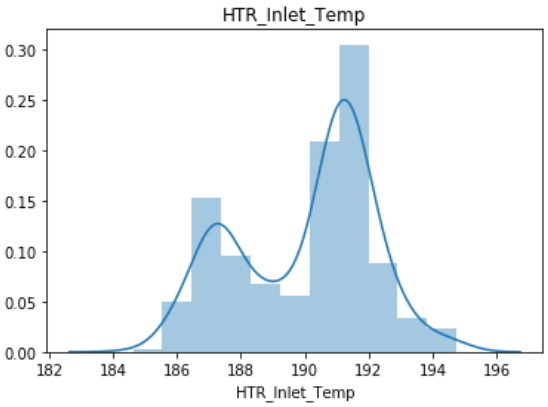

FG_LHV
FG_LHV

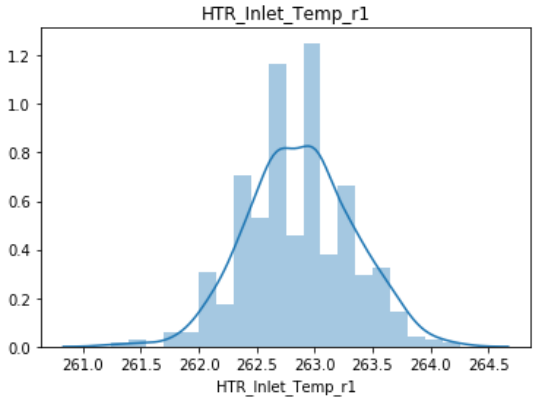

FG_LHV_r1
FG_LHV_r1

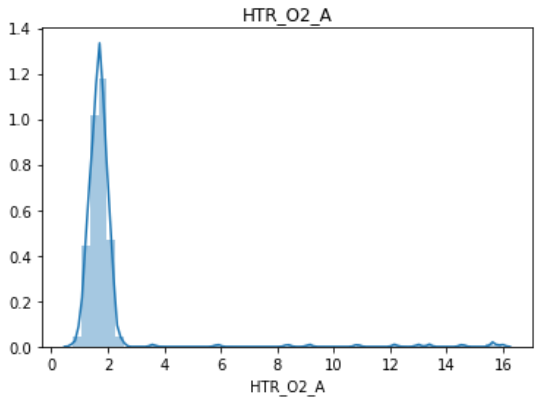

HTR_Delta_Temp
HTR_Delta_Temp

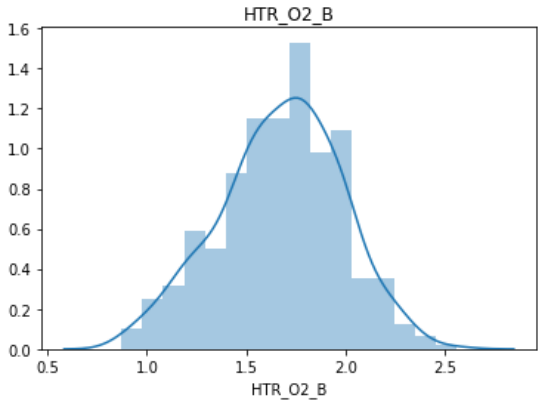

HTR_Delta_Temp_r1
HTR_Delta_Temp_r1

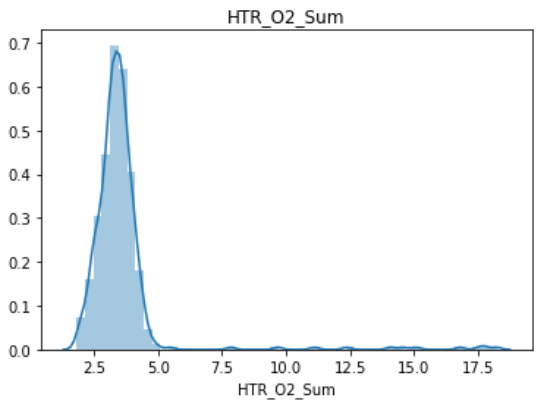

HTR_Flow
HTR_Flow

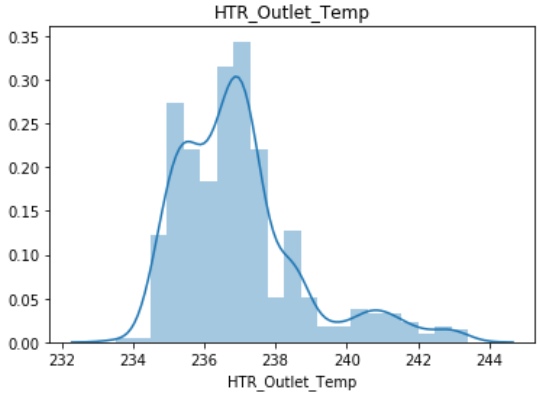

HTR_Inlet_Temp
HTR_Inlet_Temp

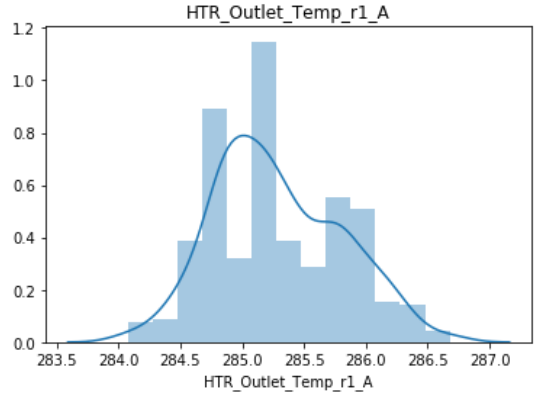

HTR_Inlet_Temp_r1
HTR_Inlet_Temp_r1

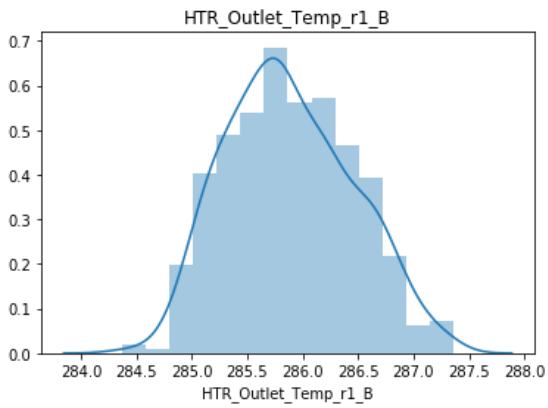

HTR_O2_A
HTR_O2_A

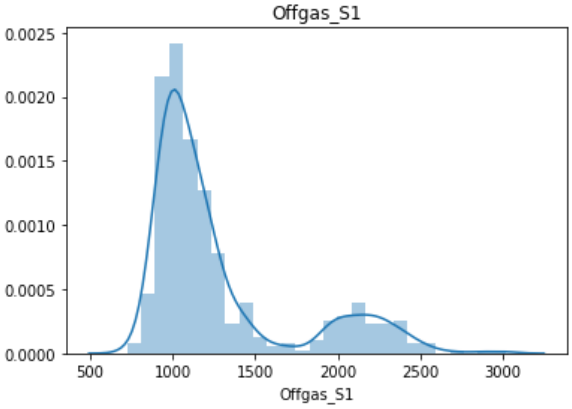

HTR_O2_B
HTR_O2_B

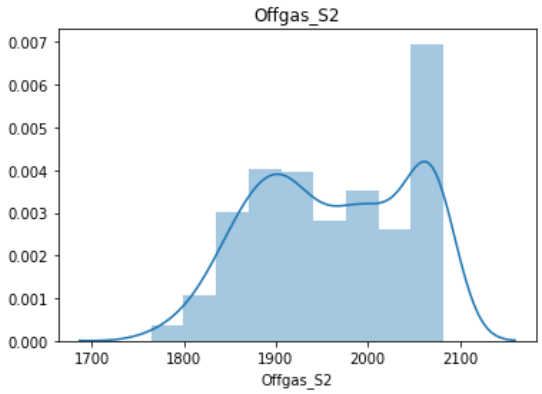

HTR_O2_Sum
HTR_O2_Sum

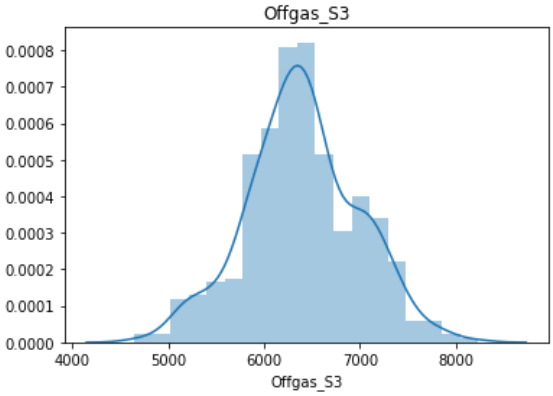

HTR_Outlet_Temp
HTR_Outlet_Temp

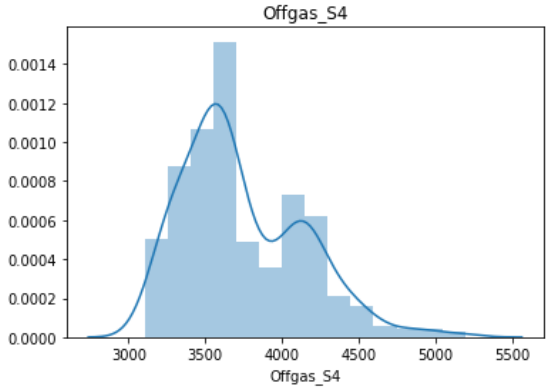

HTR_Outlet_Temp_r1_A
HTR_Outlet_Temp_r1_A

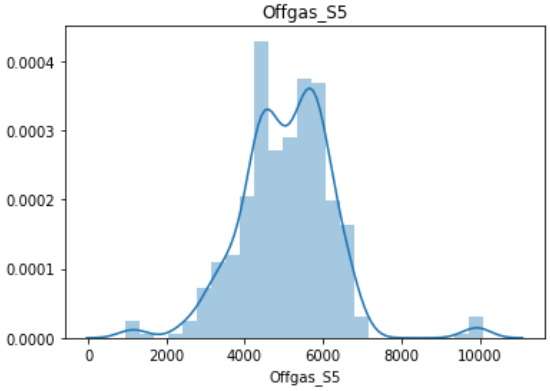

HTR_Outlet_Temp_r1_B
HTR_Outlet_Temp_r1_B

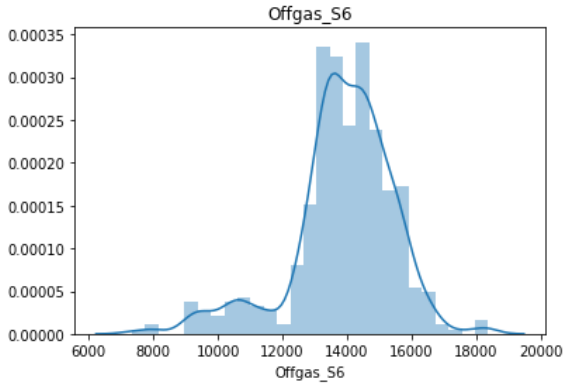

Offgas_S1
Offgas_S1



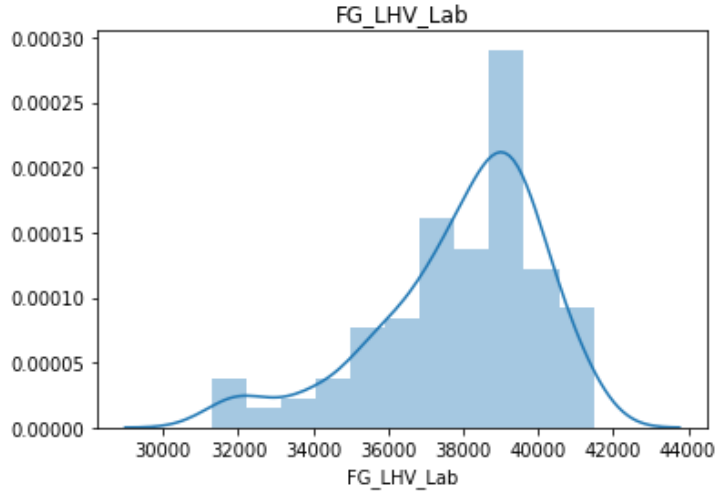




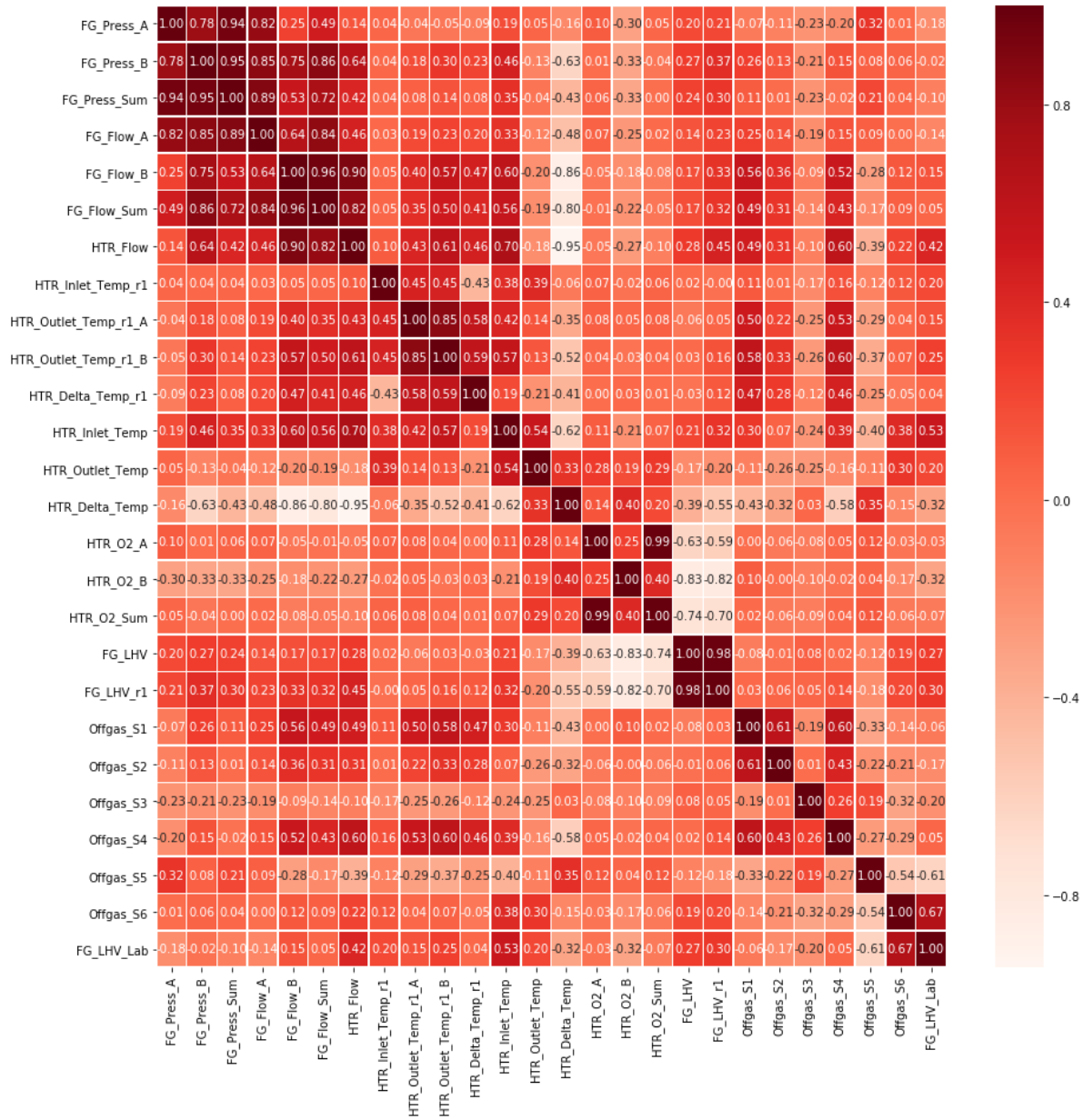




➢ Target Variable Distribution



➢ Correlation Check (via Heat map)



➢ Multicollinearity Check (Variance Inflation Factor)

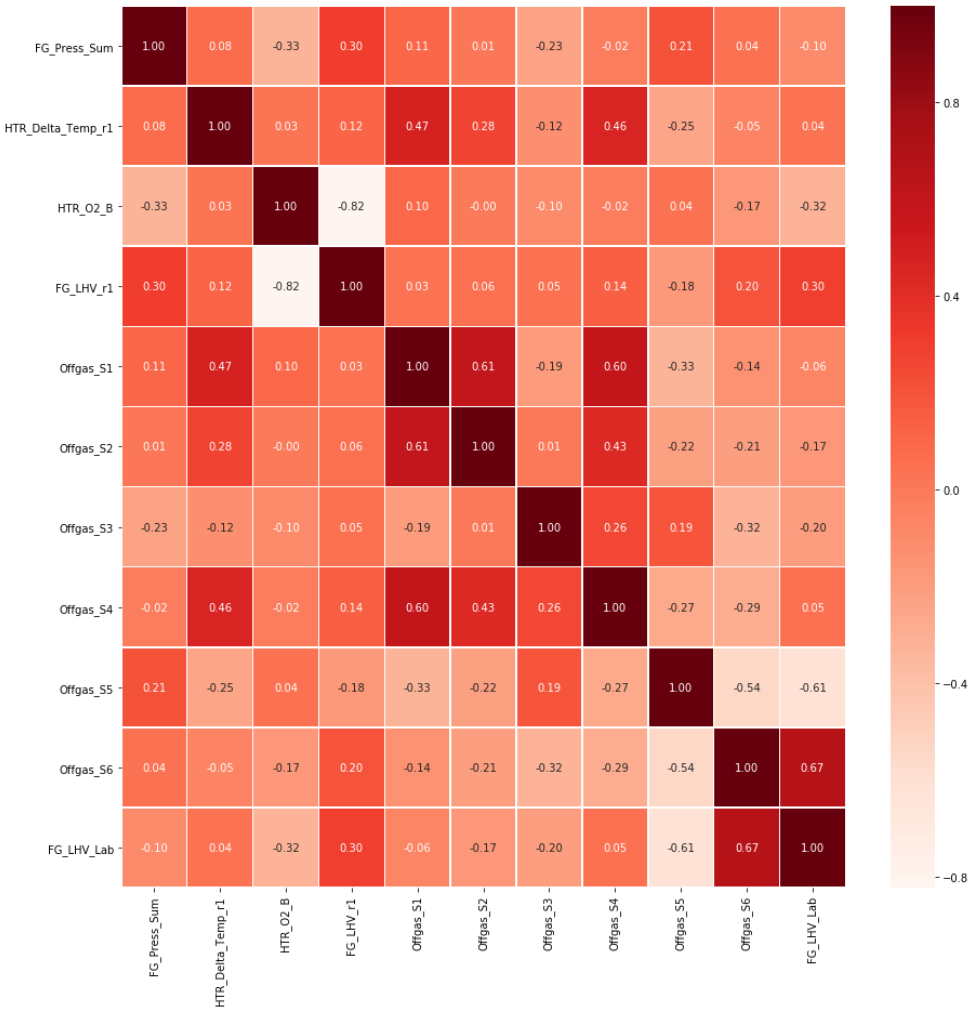
	VIF Factor	features
0	8.779651e+05	Intercept
1	3.332382e+02	df.iloc[:, :-1][0]
2	3.246305e+02	df.iloc[:, :-1][1]
3	1.041477e+03	df.iloc[:, :-1][2]
4	8.408837e+08	df.iloc[:, :-1][3]
5	2.707252e+09	df.iloc[:, :-1][4]
6	5.905222e+09	df.iloc[:, :-1][5]
7	2.792776e+01	df.iloc[:, :-1][6]
8	1.958087e+14	df.iloc[:, :-1][7]
9	5.492195e+13	df.iloc[:, :-1][8]
10	8.042142e+13	df.iloc[:, :-1][9]
11	2.196878e+14	df.iloc[:, :-1][10]
12	1.448534e+05	df.iloc[:, :-1][11]
13	1.267943e+05	df.iloc[:, :-1][12]
14	1.619995e+05	df.iloc[:, :-1][13]
15	2.448167e+05	df.iloc[:, :-1][14]
16	4.580577e+03	df.iloc[:, :-1][15]
17	2.684059e+05	df.iloc[:, :-1][16]
18	1.344784e+03	df.iloc[:, :-1][17]
19	1.255269e+03	df.iloc[:, :-1][18]
20	5.034606e+00	df.iloc[:, :-1][19]
21	2.378084e+00	df.iloc[:, :-1][20]
22	2.223714e+00	df.iloc[:, :-1][21]
23	6.852606e+00	df.iloc[:, :-1][22]
24	3.443065e+00	df.iloc[:, :-1][23]
25	5.031219e+00	df.iloc[:, :-1][24]

좌측 표와 같이, 대부분의 VIF가 상당히 크게 나왔으며 이는 독립변수 간 영향 관계가 상당히 높다고 볼 수 있다.

즉, 변수 사이에 함수 관계가 명확히 존재하기 때문에 회귀 분석 시 회귀계수의 분산이 증가하고 추정치가 불안정해진다. 통상적으로 VIF가 5 이상일 경우 다중공선성을 의심하기 때문에 위 Correlation Map과 옆 VIF 표를 참고하여 독립변수 중 일부를 제거하기로 한다.

```
# Drop Unnecessary Columns
df = df.drop(['FG_Press_A'], axis=1)
df = df.drop(['FG_Press_B'], axis=1)
df = df.drop(['FG_Flow_A'], axis=1)
df = df.drop(['FG_Flow_B'], axis=1)
df = df.drop(['HTR_Inlet_Temp'], axis=1)
df = df.drop(['HTR_Outlet_Temp'], axis=1)
df = df.drop(['HTR_Inlet_Temp_r1'], axis=1)
df = df.drop(['HTR_Outlet_Temp_r1_A'], axis=1)
df = df.drop(['HTR_Outlet_Temp_r1_B'], axis=1)
df = df.drop(['HTR_O2_A'], axis=1)
df = df.drop(['HTR_O2_Sum'], axis=1)
df = df.drop(['FG_Flow_Sum'], axis=1)
df = df.drop(['HTR_Flow'], axis=1)
df = df.drop(['HTR_Delta_Temp'], axis=1)
df = df.drop(['FG_LHV'], axis=1)
```

➢ 일부 변수 Drop 후 Correlation Check (via Heat map)



➢ 일부 변수 Drop 후 Multicollinearity Check (Variance Inflation Factor)

	VIF Factor	features
0	6221.249009	Intercept
1	1.523365	df.iloc[:, :-1][0]
2	1.398948	df.iloc[:, :-1][1]
3	3.961540	df.iloc[:, :-1][2]
4	3.454301	df.iloc[:, :-1][3]
5	3.623440	df.iloc[:, :-1][4]
6	2.174653	df.iloc[:, :-1][5]
7	1.575643	df.iloc[:, :-1][6]
8	2.984825	df.iloc[:, :-1][7]
9	3.009172	df.iloc[:, :-1][8]
10	3.680661	df.iloc[:, :-1][9]

독립변수 중 영향 관계가 높다고 판단되는 변수를 순차적으로 제거하는 시도를 진행하였으며, 최종적으로 Input 변수 25개 중 15개를 제거하였다.

최종 10개의 Input 변수에 대한 VIF 결과가 5 이하로 분석되었으며, 이로써 어느정도 다중공선성이 해소되었다고 보여 다음 분석을 진행하였다.

3. 방법론

분석에 사용된 방법론은 총 5가지이며, 각 방법론에 대한 설명과 코드는 다음과 같다.

1) Simple Linear Regression (Ordinary Least Square)

가장 보편적인 최소제곱법 회귀 분석으로, 참값과 추정값 간 오차의 제곱이 작아지게 만드는 방법이다. 현재 회사에서 생산 제품의 품질 추정을 위한 식을 구성할 때 사용하는 방법이기 때문에 예측의 신뢰성을 확인하기 용이하여 방법론으로 선정하였다.

```
# Splitting the dataset into the Training set and Test set
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)
```

```
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred_ols = regressor.predict(X_test)
print('Accuracy: %.4f' % r2_score(y_test, y_pred_ols))

# Caculating MSE
mse = np.mean((y_pred_ols - y_test)**2)
print('MSE: %.4f' % mse)

# Checking the magnitude of coefficients
predictors = X_train.columns
coef = Series(regressor.coef_, predictors).sort_values()
coef.plot(kind='bar', title='Model Coefficients')
```

Accuracy : 0.6995

MSE : 1725647.0351

2) PCA Algorithm Linear Regression (Ordinary Least Square)

주성분 분석 방법론으로, VIF 분석을 통해 다중공선성을 1차적으로 제거했지만 그래도 남아있는 상관관계를 없애고 분산을 극대화할 수 있도록 변수를 결합한 뒤 회귀 분석을 진행하였다.

```
# Fitting the PCA algorithm with our Data
pca = PCA().fit(X_train)

# Plotting the Cumulative Summation of the Explained Variance
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Dataset Explained Variance')
plt.show()
```

주성분 확인 결과, 제5주성분까지만 사용하여도 분산에 대한 설명력이 1에 가까워지기 때문에 Number of Components는 5로 설정하여 분석을 진행하였다.

```
# Applying PCA function on training
# and testing set of X component
pca = PCA(n_components = 5)

X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor_pca = LinearRegression()
regressor_pca.fit(X_train_pca, y_train)

# Predicting the Test set results
y_pred_ols_pca = regressor_pca.predict(X_test_pca)
print('Accuracy: %.4f' % r2_score(y_test, y_pred_ols_pca))

# Caculating MSE
mse = np.mean((y_pred_ols_pca - y_test)**2)
print('MSE: %.4f' % mse)
```

Accuracy : 0.6906

MSE : 1777024.6464

3) LASSO Regression (Least Absolute Shrinkage Selector Operator)

기존 OLS Modeling 시 발생할 수 있는 Overfitting 문제를 방지하기 위하여 LASSO Regression을 진행하였다. LASSO Regression은 OLS 방식에서 선형 회귀 계수에 대한 제약 조건을 추가하여 Model의 Overfitting 현상을 막는 방법론으로, 회귀 계수의 절대값의 합이 특정 값 이하이도록 만족하는 선에서 회귀 계수를 찾는 것이 목적이다. 비슷한 정규화 선형 회귀 방법인 Ridge Regression과의 가장 큰 차이점은 회귀 계수를 0으로 만들 수 있다는 점이다. (Ridge의 경우, Model 신뢰성에 악영향을 주는 계수에 Weight을 작게 줄 수 있지만, 0으로 만들지는 못 한다.)

Feature Selection 시 중요하게 작용하는 Alpha값은 여러 번의 시도로 최적 값을 찾아냈으며, 코드는 아래와 같다.

```
# Lasso Regression (Least Absolute Shrinkage Selector Operator)
from sklearn.linear_model import Lasso

lassoReg = Lasso(alpha= 0.6, normalize = True)
lassoReg.fit(X_train, y_train)

pred = lassoReg.predict(X_test)

# Predicting the Test set results
y_pred_lasso = pred
print('Accuracy: %.4f' % r2_score(y_test, y_pred_lasso))

# Caculating MSE
mse = np.mean((y_pred_lasso - y_test)**2)
print('MSE: %.4f' % mse)

# Checking the magnitude of coefficients
predictors = X_train.columns
coef = Series(lassoReg.coef_, predictors).sort_values()
coef.plot(kind='bar', title='Model Coefficients')
```

Accuracy : 0.7004

MSE : 1720684.9346

4) Elastic Net Regression

Elastic Net Regression은 Ridge Regression과 Lasso Regression의 페널티 항을 모두 추가한 방법론이다. 위에서 Lasso Regression 분석 시 중요하다고 판단된 Feature만 채택되고, 나머지 Feature에 대한 정보가 손실됨에 따라 Model의 정확도가 떨어질 수 있는 점을 보완하기 위해 Ridge와 Lasso의 하이브리드 형태인 Elastic Net 분석을 시도하였다.

Elastic Net의 주요 Parameter인 Alpha값(L1 term, L2 term의 합)과 L1_ratio(L1 term에 대한 가중치)을 바꿔가며 여러 시도 끝에 가장 최적 값을 찾아내었으며, 코드는 아래와 같다.

```
# Elastic Net Regression
from sklearn.linear_model import ElasticNet

ENReg = ElasticNet(alpha = 0.01, l1_ratio = 0.7, normalize = False)
ENReg.fit(X_train, y_train)

pred = ENReg.predict(X_test)

# Predicting the Test set results
y_pred_EN = pred
print('Accuracy: %.4f' % r2_score(y_test, y_pred_EN))

# Caculating MSE
mse = np.mean((y_pred_EN - y_test)**2)
print('MSE: %.4f' % mse)

# Checking the magnitude of coefficients
predictors = X_train.columns
coef = Series(ENReg.coef_, predictors).sort_values()
coef.plot(kind='bar', title='Model Coefficients')
```

Accuracy : 0.7012

MSE : 1716106.2515

5) XGBoost Regression

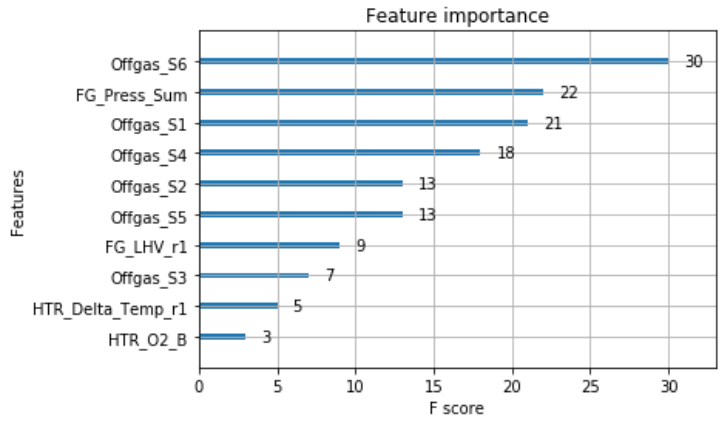
XGBoost는 Gradient Boosting 알고리즘 기반의 앙상블 모델로 순차적 학습/예측 과정을 병렬 처리가 가능하도록 구현한 방법론이다. (Tree Building Step에서 병렬 처리가 가능함.) XGB는 CART(Classification and Regression Tree)를 기반으로 하기 때문에 본 프로젝트의 회귀 분석에 XGB Regressor를 적용하였다. XGBRegressor의 경우, Tuning parameter가 상당히 많아 적당한 Tuning 값을 찾기가 어려웠으나 향후 Hyperparameter Tuning 기법 (예: Bayesian Optimization)을 통해 Parameter 최적화를 시도해보고자 한다. XGB Regression 코드는 아래와 같다.

```
# XGBoost Regression
from xgboost import XGBRegressor
from xgboost import plot_importance

XGB = XGBRegressor(silent=False,
                   booster='gbtree',
                   min_child_weight = 8, # default = 1, 높을수록 under-fitting
                   max_depth=3, # default = 6, typical 3 ~ 10
                   colsample_bytree = 0.9, # 트리 생성 시 훈련 데이터 변수 샘플링 비율, 보통 0.6 ~ 0.9
                   colsample_bylevel = 0.9, # 트리의 레벨 별로 훈련 데이터 변수 샘플링 비율, 보통 0.6 ~ 0.9
                   subsample = 0.499,
                   objective='reg:linear',
                   eval_metric = 'rmse',
                   n_estimators=61,
                   seed=100)

XGB.fit(X_train, y_train)
pred = XGB.predict(X_test)

# Predicting the Test set results
y_pred_XGB = pred
print('Accuracy: %.4f' % r2_score(y_test, y_pred_XGB))

# Caculating MSE
mse = np.mean((y_pred_XGB - y_test)**2)
print('MSE: %.4f' % mse)

# Checking the feature importance
plot_importance(XGB)
```

Accuracy : 0.7114

MSE : 1657462.8988

아래는 XGB Regressor를 통해 분석된 Feature Importance 결과로 Offgas_S6 및 FG_Press_Sum 등 실제로 Fuel Gas 열량에 가장 영향이 큰 변수를 적절히 찾았음을 알 수 있다.

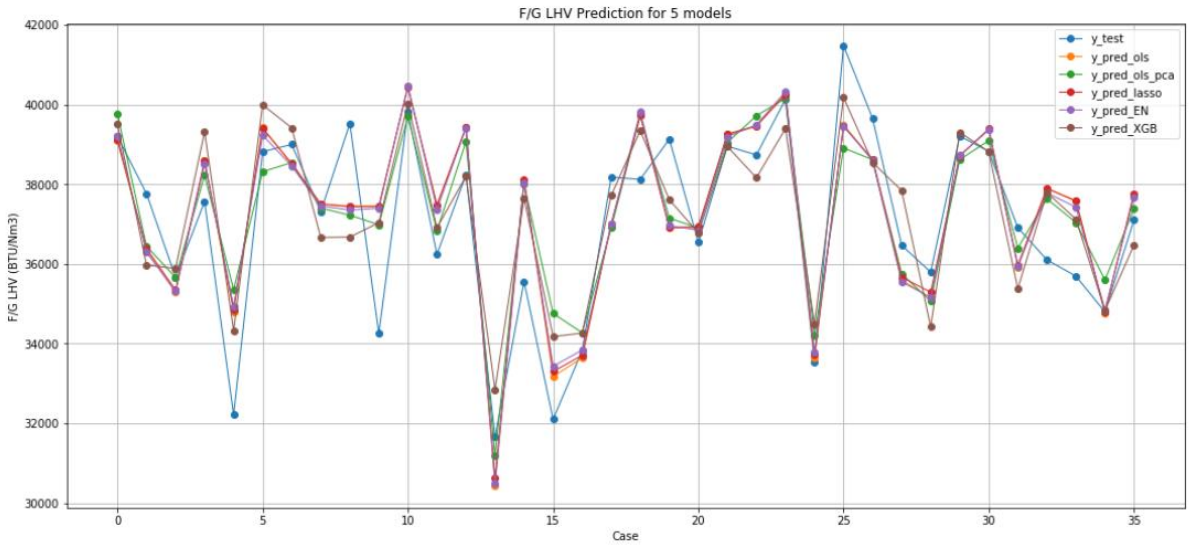


4. 분석 결과 및 해석

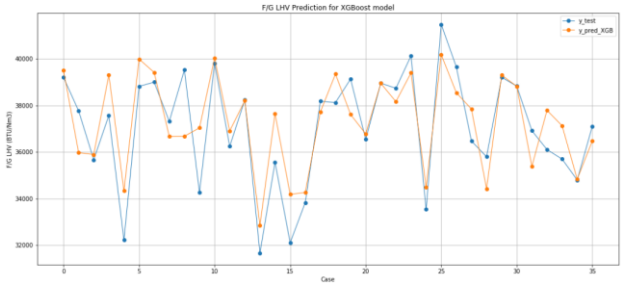
위 5가지 회귀 분석 (Simple Linear Regression, PCA, LASSO, Elastic Net, XGBoost) 결과에 대해 아래와 같이 일정 오차 범위 (평균 열량 값의 +-5% 수준인 2,200 BTU/Nm3) 내 예측 성능을 평가하였으며, XGBoost 기법의 회귀 Model 신뢰성이 가장 높은 것으로 나타났다.

구분	Simple LR	PCA Reg.	LASSO	ElasticNet	XGB Reg.
Test Data 개수	36	36	36	36	**36**
오차범위 내 예측 개수	32	30	32	33	**34**
오차범위 밖 예측 개수	4	6	4	3	**2**
오차범위 미만 예측률	88.89%	83.33%	88.89%	91.67%	**94.44%**
오차 평균 절대값	1049.1	1018.7	1037.9	1036.0	**1029.1**
Model Accuracy	0.6995	0.6906	0.7004	0.7012	**0.7114**
순위	4	5	3	2	**1**

각 Model의 예측 결과와 Test Data를 비교한 Trend는 다음과 같다.



5가지의 Model이 전반적으로 대동소이한 Trend를 보이며, 이는 Model 방법론에 따라 예측 값의 차이가 존재하나 방향성은 거의 동일한 것으로 판단된다. (일부 구간 제외)



* 좌측 Trend는 가장 높은 정확도를 보인 XGB Regression과 Test Data의 Trend를 비교함.

5. 한계점 및 향후 계획

1) 한계점

- 예측 Target Data에 필요한 Fuel Gas의 조성 분석이 보통 주 1회 실시함에 따라 분석을 위한 Data가 충분하지 않았음.
- Fuel Gas 조성 분석 시간을 보통 05:00 AM 실시하여 Input 변수 Data 취득 시 05:00 AM 기준으로 취득하였으나, 실제 Fuel Gas 조성 분석 시간이 일정하지 않을 수 있음. (+- 30분~1시간 정도의 차이가 발생하기도 함)
- 분석에 사용한 Data의 절대적 양이 적기 때문에 대표성 없는 일부 데이터로 인한 Sampling noise가 발생할 수 있고, Model 신뢰성에 심각한 악영향을 줄 수 있음.
- XGB Regression 뿐 아니라 LASSO, Elastic Net 등 다양한 Parameter Tuning 시 Trial-Error를 통해 Manual로 Tuning하여 최적값을 찾지 못 하였을 수 있음. 향후 Hyperparameter Tuning 기법(예: Bayesian Optimization) 등을 통해 보다 효율적이고 정확한 Tuning을 시도해보고자 함.

2) 향후 계획

- 현재 본인이 담당하고 있는 공정자동화 시스템인 APC (Advanced Process Control System)에 위 분석 Model 적용을 시도하고자 하며, 계획은 아래와 같음. (APC Server 부하 고려하여 5가지 Model 선택적 적용 예정)
  (1) 공장 내 Control Room에 위치한 APC Server 내 Python Code를 단순화하여 구동함.
  (2) 실시간 공정 값(from DCS)을 APC Server 내 Excel로 취득함.
  (3) Excel 내 Input 변수 값을 기반으로 실시간 예측 값을 다시 Excel로 변환/저장함.
  (4) APC 제어 시 해당 예측 값을 사용하여 Fuel Gas Control 시 반영함.
- 단, 위 적용 전 신규 Data를 통해 Model 신뢰성을 향상시키고 잘못된 예측 값에 의한 공정 제어 에러 발생을 방지하기 위한 추가적인 안정 장치(Safety Interlock) 구축을 검토할 예정임.

별첨 1. 데이터 (FG_LHV_r1.csv)

별첨 2. 코드 (FG_LHV_r1.py)