SCALE 불량 요인 분석 및 개선안 도출

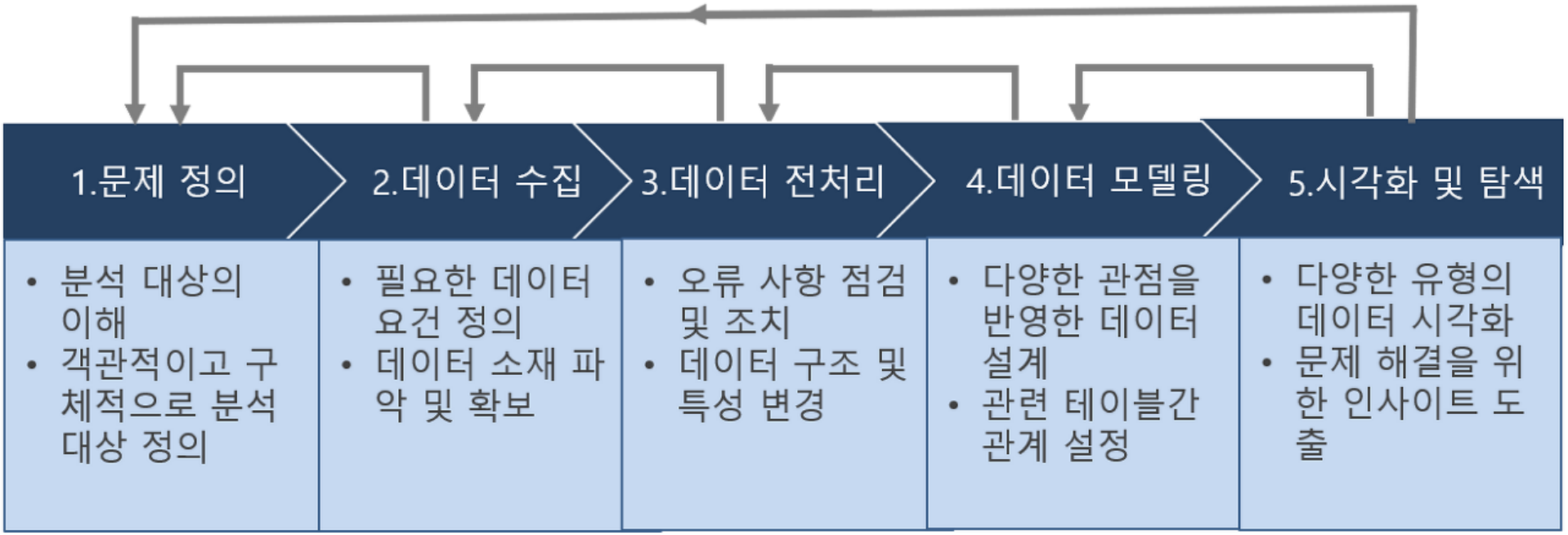
POSCO AI BIGDATA 아카데미 종합실습 2

A반 이무동

과제 정의

강판 제조공정 도중 불량품을 생산함으로써, 불량 자재를 유통할 가능성 有
불량 자재 유통에 사고 끊이지 않아, 불량 철강재 생산 문제 해결방안이 시급함
'SCALE불량'과 같은 데이터를 통해 불량 철강재와 관련된 데이터 분석 진행

분석 계획



데이터 현황

```
# 데이터 구성하기
df_raw = pd.read_csv('/content/sample_data/SCALE불량.csv', encoding='euc-kr')
df_raw
```

'SCALE불량.CSV'를 READ 하여 데이터 구성하기

```
# 데이터 구성하기 - 결측치 처리
df_raw.isnull().sum()
```

```
PLATE_NO            0
ROLLING_DATE        0
SCALE               0
SPEC                0
STEEL_KIND          0
PT_THICK            0
PT_WIDTH            0
PT_LENGTH           0
PT_WEIGHT           0
FUR_NO              0
FUR_NO_ROW          0
FUR_HZ_TEMP         0
FUR_HZ_TIME         0
FUR_SZ_TEMP         0
FUR_SZ_TIME         0
FUR_TIME            0
FUR_EXTEMP          0
ROLLING_TEMP_T5     0
HSB                 0
ROLLING_DESCALING   0
WORK_GR             0
dtype: int64
```

결측치 존재하지 않음

```
# 변수 타입 파악
df_raw.dtypes
```

```
PLATE_NO            object
ROLLING_DATE        object
SCALE               object
SPEC                object
STEEL_KIND          object
PT_THICK           float64
PT_WIDTH             int64
PT_LENGTH            int64
PT_WEIGHT            int64
FUR_NO              object
FUR_NO_ROW          object
FUR_HZ_TEMP          int64
FUR_HZ_TIME          int64
FUR_SZ_TEMP          int64
FUR_SZ_TIME          int64
FUR_TIME             int64
FUR_EXTEMP           int64
ROLLING_TEMP_T5      int64
HSB                 object
ROLLING_DESCALING    int64
WORK_GR             object
dtype: object
```

PLATE_NO, ROLLING_DATE, SCALE, SPEC, STEEL_KIND, FUR_NO, FUR_NO_ROW, HSB, WORK_GR는 범주형 변수

각 범주형 변수에 특성을 고려하여, 각각 다른 방식으로 처리 필요

데이터 현황

SCALE

```
[ ]  # SCALE 에서 '양품'이면 0, '불량' 이면 1로 데이터 변환
     df_raw['SCALE'].replace({'양품':0, '불량':1}, inplace=True)
```

```
[ ]  # SCALE 변환 결과 간략히 확인
     df_raw['SCALE']
```

```
0      0
1      0
2      0
3      0
4      0
      ..
715    1
716    0
717    0
718    0
719    0
Name: SCALE, Length: 720, dtype: int64
```

SCALE에서 '양품' 이면 0, '불량' 이면 1로 데이터를 변환

SPEC

```
# SPEC은 앞 쪽 영문 2글자만 추출하여 새로운 SPEC_SIMPLE 변수 생성
# SPEC 간략히 확인
df_raw['SPEC']
```

```
0      AB/EH32-TM
1      AB/EH32-TM
2       NV-E36-TM
3       NV-E36-TM
4      BV-EH36-TM
          ...
715        NK-KA
716       NV-A32
717       NV-A32
718         LR-A
719       GL-A32
Name: SPEC, Length: 720, dtype: object
```

```
# SPEC 의 앞 쪽 영문 2글자 추출
df_raw['SPEC'].str[:2]
```

```
0      AB
1      AB
2      NV
3      NV
4      BV
      ..
715    NK
716    NV
717    NV
718    LR
719    GL
Name: SPEC, Length: 720, dtype: object
```

SPEC은 앞 쪽 영문 2글자만 추출하여 새로운 SPEC_SIMPLE 변수 생성

데이터 현황

FUR_NO

```
[ ] # FUR_NO는 가열로 호기로, 이를 1,2,3 과 같은 int 형태로 바꾼다면 분석값에 혼란을 끼칠수 있으므로 따로 처리하지 않는다.
    df_raw['FUR_NO']
```

```
0      1호기
1      1호기
2      2호기
3      2호기
4      3호기
      ...
715    3호기
716    2호기
717    2호기
718    3호기
719    3호기
Name: FUR_NO, Length: 720, dtype: object
```

STEEL_KIND

```
[ ] # STEEL_KIND는 강종으로 이미 분류하기 편한 상태이므로 따로 처리하지 않는다.
    df_raw['STEEL_KIND']
```

```
0      T1
1      T1
2      T8
3      T8
4      T8
       ..
715    C0
716    C0
717    C0
718    C0
719    C0
Name: STEEL_KIND, Length: 720, dtype: object
```

FUR_NO_ROW

```
[ ] # FUR_NO_ROW는 가열로 작업순번으로, 이를 1,2,3 과 같은 int 형태로 바꾼다면 분석값에 혼란을 끼칠수 있으므로 따로 처리하지 않는다.
    df_raw['FUR_NO_ROW']
```

```
0      1열
1      2열
2      1열
3      2열
4      1열
       ..
715    1열
716    1열
717    2열
718    2열
719    1열
Name: FUR_NO_ROW, Length: 720, dtype: object
```

STEEL_KIND는 강종으로 이미 분류하기 편한 상태이므로 따로 처리하지 않는다. 또한, FUR_NO, FUR_NO_ROW은 각각 가열로 호기, 가열로 작업순번이므로 이를 1, 2, 3 과 같은 int 형태로 바꾼다면 분석값에 혼란을 끼칠 수 있으므로 따로 처리하지 않는다.

데이터 현황

HSB

```
[ ]  # HSB는 Hot Scale Breaker으로 '미적용'이면 0, '적용' 이면 1로 데이터 변환
     df_raw['HSB'].replace({'미적용':0, '적용':1}, inplace=True)
```

```
[ ]  # HSB 변환 결과 간략히 확인
     df_raw['HSB']
```

```
0      1
1      1
2      1
3      1
4      1
      ..
715    1
716    1
717    1
718    1
719    1
Name: HSB, Length: 720, dtype: int64
```

HSB는 Hot Scale Breaker으로 '미적용'이면 0, '적용 ' 이면 1로 데이터 변환한다.
WORK_GR은 작업조이며, 이를 1,2,3, 과 같은 int 형태로 바꾼다면 분석값에 혼란을 끼칠수 있으므로 따로 처리하지 않는다.

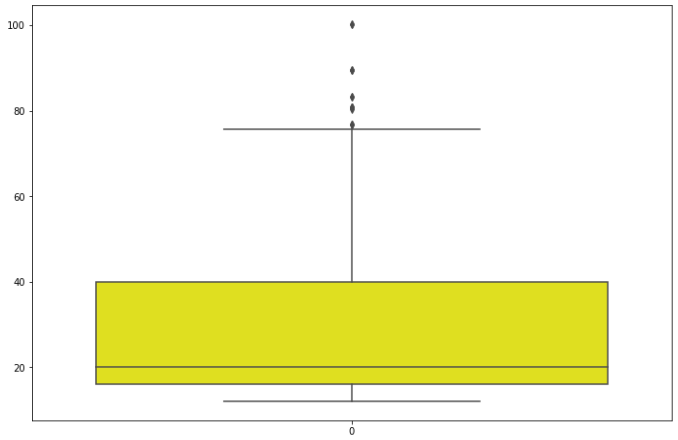
WORK_GR

```
[ ]  # WORK_GR은 작업조 이며, 이를 1,2,3 과 같은 int 형태로 바꾼다면 분석값에 혼란을 끼칠수 있으므로 따로 처리하지 않는다.
     df_raw['WORK_GR']
```

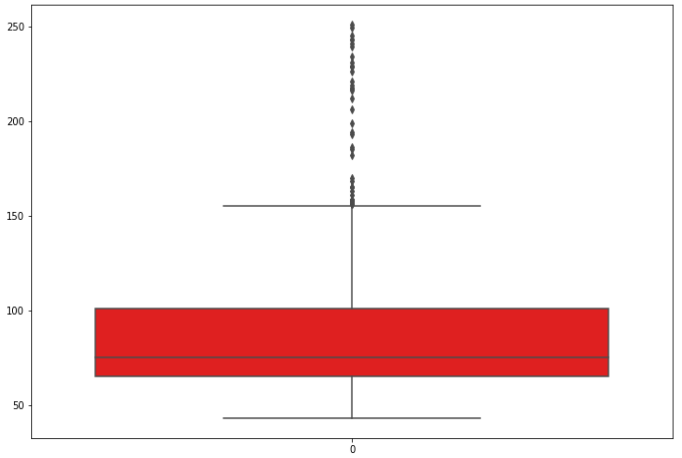
```
0      2조
1      2조
2      3조
3      3조
4      1조
       ..
715    2조
716    1조
717    4조
718    2조
719    2조
Name: WORK_GR, Length: 720, dtype: object
```

데이터 현황

```
# 이상치 제거
plt.figure(figsize=(12,8))
sns.boxplot(data=df_raw['PT_THICK'], color='yellow')
plt.show()
```

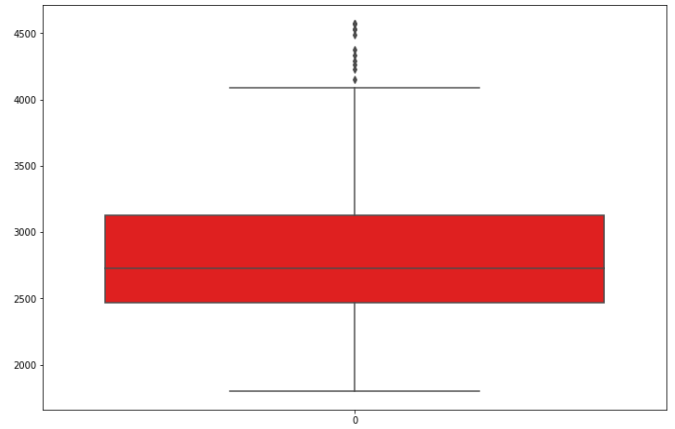



```
plt.figure(figsize=(12,8))
sns.boxplot(data=df_raw['FUR_HZ_TIME'], color='red')
plt.show()
```

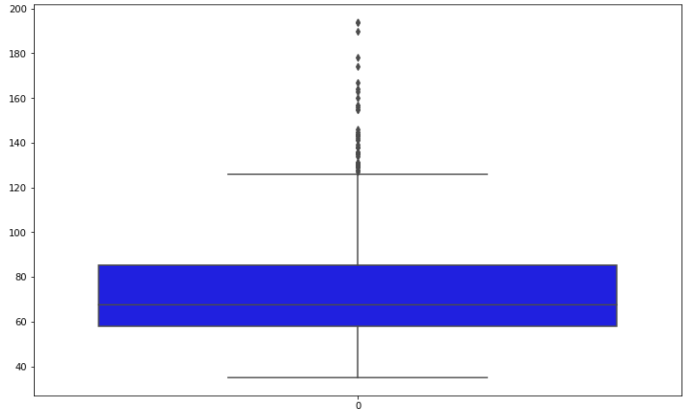


변수의 이상치 제거를 위하여 plt.figure와 sns.boxplot 을 이용하여 이상치를 확인한다. 해당 분석 결과, [PT_THICK, PT_WIDTH, FUR_HZ_TIME, FUR_SZ_TIME, ROLLING_TEMP_T5]의 이상치를 제거한다.

```
plt.figure(figsize=(12,8))
sns.boxplot(data=df_raw['PT_WIDTH'], color='red')
plt.show()
```




```
plt.figure(figsize=(12,8))
sns.boxplot(data=df_raw['FUR_SZ_TIME'], color='blue')
plt.show()
```



최종 DATA -> 목표변수 : SCALE(Scale 불량)

```
df_raw.isnull().sum()
# 이상치 제거
df_raw.dropna(inplace=True)
df_raw
```

	PLATE_NO	ROLLING_DATE	SCALE	SPEC	STEEL_KIND	PT_THICK	PT_WIDTH	PT_LENGTH	PT_WEIGHT	FUR_NO	...	FUR_HZ_TIME	FUR_SZ_TEMP	FUR_SZ_TIME	FUR_TIME	FUR_EXTEMP	ROLLING_TEMP_T5	HSB
0	PB562774	2021-08-01:08:00:01	0	AB/EH32-TM	T1	32.25	3707.0	15109	14180	1호기	...	116.0	1133	59.0	282	1125	934.0	1
1	PB562775	2021-08-01:08:07:11	0	AB/EH32-TM	T1	32.25	3707.0	15109	14180	1호기	...	122.0	1135	53.0	283	1120	937.0	1
2	PB562776	2021-08-01:08:14:21	0	NV-E36-TM	T8	33.27	3619.0	19181	18130	2호기	...	116.0	1121	55.0	282	1106	889.0	1
3	PB562777	2021-08-01:08:21:31	0	NV-E36-TM	T8	33.27	3619.0	19181	18130	2호기	...	125.0	1127	68.0	316	1113	885.0	1
4	PB562778	2021-08-01:08:28:41	0	BV-EH36-TM	T8	38.33	3098.0	13334	12430	3호기	...	134.0	1128	48.0	314	1118	873.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
715	PB563502	2021-08-04:21:24:11	1	NK-KA	C0	20.14	3580.0	38639	21870	3호기	...	72.0	1164	62.0	245	1155	1005.0	1
716	PB563503	2021-08-04:21:31:21	0	NV-A32	C0	15.08	3212.0	48233	18340	2호기	...	61.0	1169	61.0	238	1160	947.0	1
717	PB563504	2021-08-04:21:38:31	0	NV-A32	C0	16.60	3441.0	43688	19590	2호기	...	65.0	1163	77.0	247	1152	948.0	1
718	PB563505	2021-08-04:21:45:41	0	LR-A	C0	15.59	3363.0	48740	80240	3호기	...	86.0	1163	45.0	243	1154	940.0	1
719	PB563506	2021-08-04:21:52:51	0	GL-A32	C0	16.09	3400.0	54209	69840	3호기	...	82.0	1169	45.0	239	1155	957.0	1

609 rows × 22 columns

Normalizer 적용을 통한 Heatmap 출력

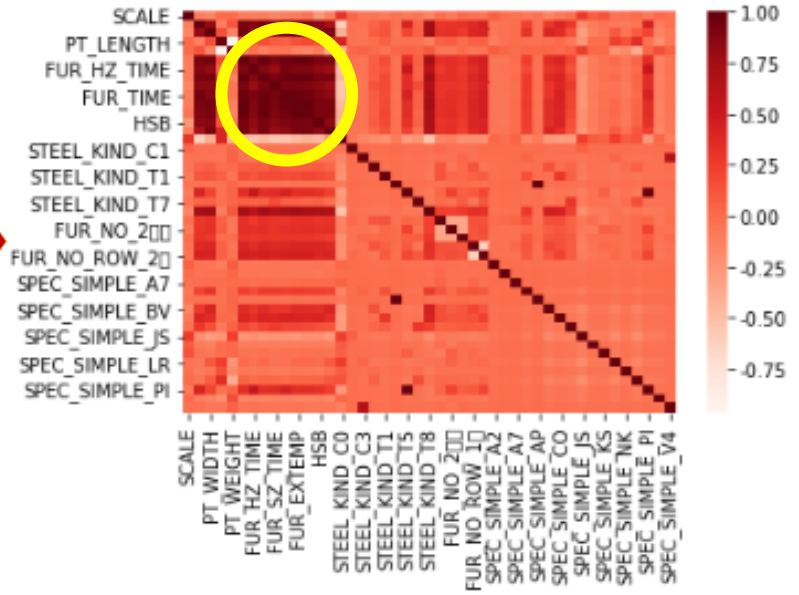
```
# Normalizer 적용
from sklearn.preprocessing import Normalizer
```

```
# Scale 변환 : Normalizer scaler (평균, 표준편차 적용)
df_scale_normal = Normalizer()
df_scale_normal = df_scale_normal.fit_transform(df_raw_dummy)
df_scale_normal
```

```
# Scale 변환 결과값의 전체 상관관계 분석
df_scale_normal.corr().round(3)
```

	SCALE	PT_THICK	PT_WIDTH	PT_LENGTH	PT_WEIGHT	FUR_HZ_TEMP	FUR_HZ_TIME	FUR_SZ_TEMP	FUR_SZ_TIME	FUR_TIME
SCALE	1.000	-0.101	-0.136	0.013	0.011	-0.088	-0.099	-0.087	-0.144	-0.102
PT_THICK	-0.101	1.000	0.811	-0.112	0.135	0.917	0.884	0.919	0.858	0.949
PT_WIDTH	-0.136	0.811	1.000	0.162	-0.128	0.944	0.812	0.944	0.853	0.879
PT_LENGTH	0.013	-0.112	0.162	1.000	-0.964	0.165	0.047	0.163	0.092	0.079
PT_WEIGHT	0.011	0.135	-0.128	-0.964	1.000	-0.122	-0.017	-0.120	-0.054	-0.045
FUR_HZ_TEMP	-0.088	0.917	0.944	0.165	-0.122	1.000	0.895	1.000	0.918	0.971
FUR_HZ_TIME	-0.099	0.884	0.812	0.047	-0.017	0.895	1.000	0.893	0.801	0.926
FUR_SZ_TEMP	-0.087	0.919	0.944	0.163	-0.120	1.000	0.893	1.000	0.916	0.971
FUR_SZ_TIME	-0.144	0.858	0.853	0.092	-0.054	0.918	0.801	0.916	1.000	0.925
FUR_TIME	-0.102	0.949	0.879	0.079	-0.045	0.971	0.926	0.971	0.925	1.000
FUR_EXTEMP	-0.087	0.919	0.944	0.163	-0.120	1.000	0.893	1.000	0.916	0.971
ROLLING_TEMP_T5	-0.042	0.897	0.933	0.185	-0.137	0.994	0.885	0.994	0.904	0.961
HSB	-0.253	0.852	0.870	0.128	-0.090	0.928	0.834	0.929	0.860	0.905
ROLLING_DESCALING	-0.105	0.730	0.919	0.412	-0.361	0.923	0.778	0.922	0.795	0.841
STEEL_KIND_C0	0.325	-0.608	-0.403	0.426	-0.437	-0.468	-0.519	-0.465	-0.544	-0.536
STEEL_KIND_C1	0.034	-0.021	-0.035	-0.037	0.034	-0.028	-0.016	-0.028	-0.031	-0.023
STEEL_KIND_C3	-0.048	-0.035	-0.045	-0.050	0.059	-0.045	-0.050	-0.043	-0.058	-0.049
STEEL_KIND_T0	-0.047	0.055	0.080	-0.004	0.013	0.079	0.087	0.076	0.137	0.087
STEEL_KIND_T1	-0.080	0.120	0.137	0.067	-0.044	0.178	0.129	0.180	0.191	0.174
STEEL_KIND_T3	-0.036	-0.023	-0.042	-0.033	0.035	-0.031	-0.034	-0.032	-0.029	-0.037

```
#  Scale 변환 결과값의 Heatmap 출력
sns.heatmap(df_scale_normal.corr(), cmap="Reds")
```

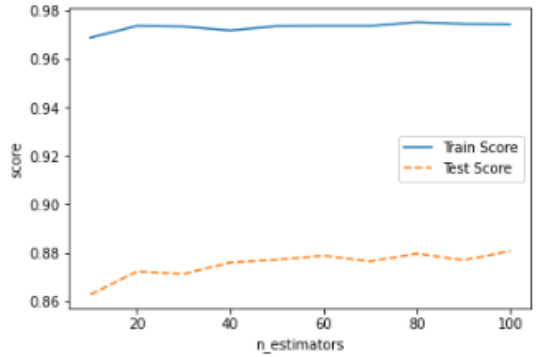


Scale 변환 결과값의 Heatmap 출력 결과, FUR_HZ_TIME, FUR_TIME, HSB 와 SCALE이 연관이 있다는걸 알 수 있다.

Random Forest

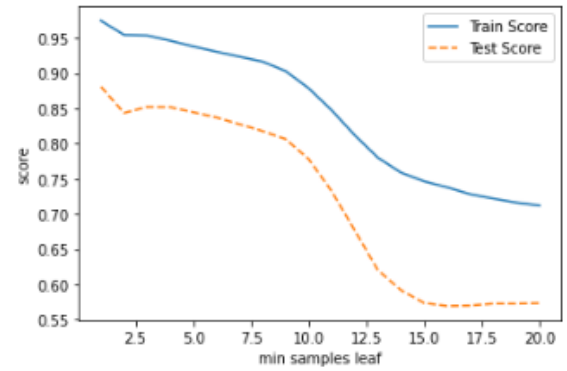
```
# 모델 설명력에 대한 그래프 확인
plt.plot(para_n_tree, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_n_tree, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score");plt.xlabel("n_estimators")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22be79190>
```



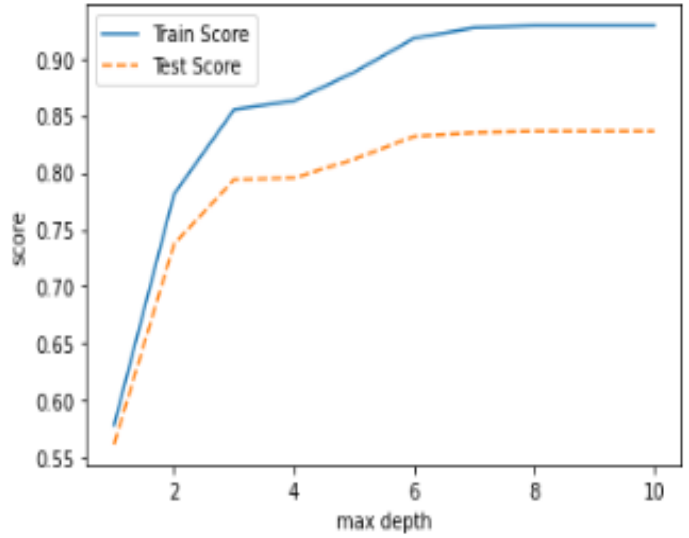

```
# 모델 설명력에 대한 그래프 확인
plt.plot(para_leaf, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_leaf, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score");plt.xlabel("min samples leaf")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22c99b490>
```




```
# 모델 설명력에 대한 그래프 확인
plt.plot(para_depth, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_depth, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score");plt.xlabel("max depth")
plt.legend()
```

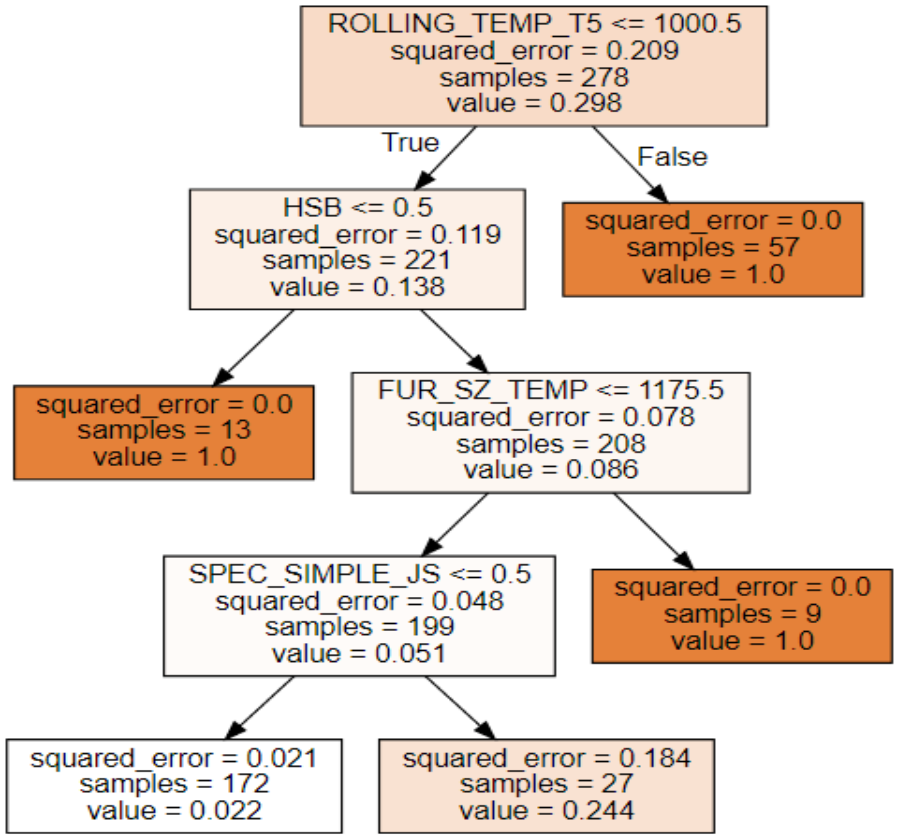
```
<matplotlib.legend.Legend at 0x7fa22c046b90>
```



모델 설명력에 대한 여러 그래프의 확인 내용.
y_label 은 'score'로 동일하지만, x_label은 'n_estimators' , 'min samples leaf', 'max depth'로 세 가지의 다른 그래프들을 확인할 수 있다.

Random Forest 트리 및 설명변수 중요도

```
#10번 트리
export_graphviz(rf_final.estimators_[10], out_file="rfr_final_10.dot", feature_names= v_feature_name, impurity = True, filled = True)

#tree_final_10.dot 그리기
with open('rfr_final_10.dot')as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```

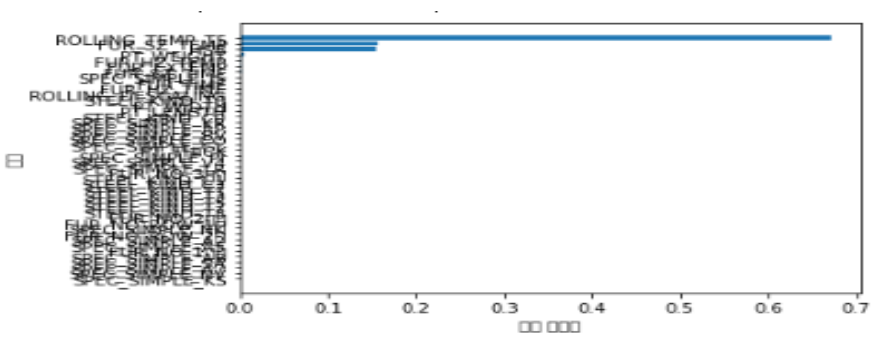



```
# tree.feature_importances_로 설명변수 중요도 확인 및 테이블로 저장
df_importance = pd.DataFrame()
df_importance["Feature"] = v_feature_name
df_importance["Importance"] = rf_final.feature_importances_

#df_feature_importance의 테이브를 중요도 순으로 정렬
df_importance.sort_values("Importance", ascending = False, inplace = True)
df_importance.round(3)
```

	Feature	Importance
10	ROLLING_TEMP_T5	0.672
6	FUR_SZ_TEMP	0.156
11	HSB	0.155
3	PT_WEIGHT	0.004
4	FUR_HZ_TEMP	0.004
9	FUR_EXTEMP	0.003
7	FUR_SZ_TIME	0.002
35	SPEC_SIMPLE_JS	0.001
8	FUR_TIME	0.001
5	FUR_HZ_TIME	0.000

'ROLLING_TEMP_T5'의 Importance는 0.672, **'FUR_SZ_TEMP'**의 Importance는 0.156, **'HSB'**의 Importance는 0.155로 세 가지 설명변수가 SCALE에 가장 영향을 많이 주는 변수임을 알 수 있다

```
#설명변수 중요도 그래프
#중요도가 높은 변수를 상위에 그림.
df_importance.sort_values("Importance", ascending=True, inplace=True)
coordinates = range(len(df_importance))
plt.barh(y=coordinates, width = df_importance["Importance"])
plt.yticks(coordinates, df_importance["Feature"])
plt.xlabel("변수 중요도")
plt.ylabel("변수")
```

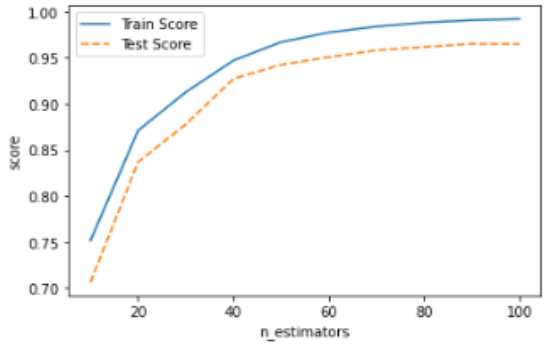


압연온도(ROLLING_TEMP_T5). 가열로 균열대 시간(FUR_SZ_TEMP), 고열의 스케일 브레이커(HSB) 순으로 SCALE과 강한 연관성이 있음

Gradient Boosting

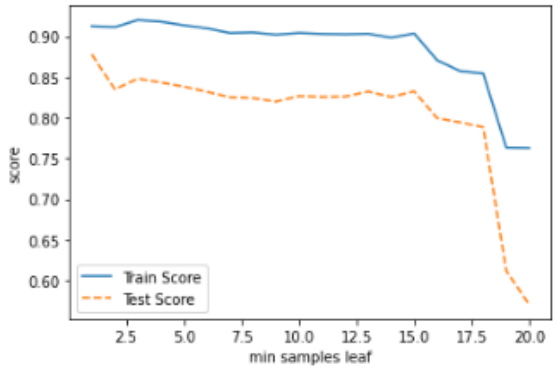
```
# 모델 설명력에 대한 그래프 확인
plt.plot(para_n_tree, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_n_tree, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score"); plt.xlabel("n_estimators")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22c45c2d0>
```



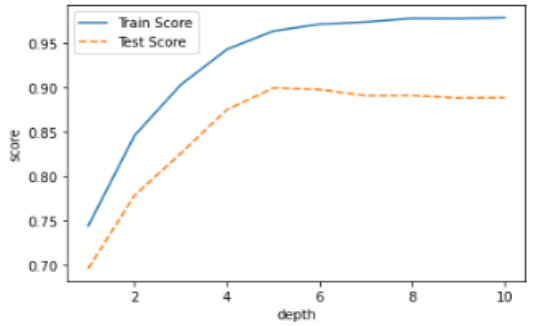

```
# 모델 설명력에 대한 그래프 확인
plt.plot(para_leaf, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_leaf, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score"); plt.xlabel("min samples leaf")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22c3e96d0>
```



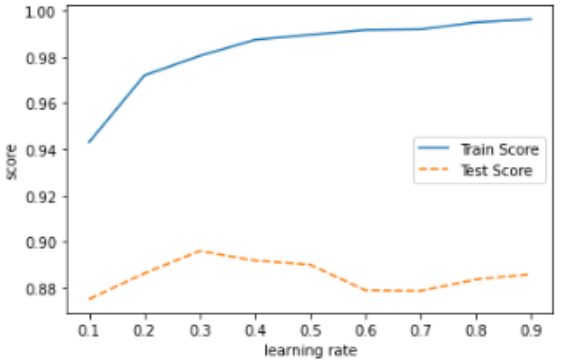

```
# 모델 설명력에 대한 그래프 확인 : 22개
plt.plot(para_depth, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_depth, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score")
plt.xlabel("depth")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22aa27ed0>
```




```
# 모델 설명력에 대한 그래프 확인 : 22개
plt.plot(para_lr, train_score, linestyle = "-", label = "Train Score")
plt.plot(para_lr, test_score, linestyle = "--", label = "Test Score")
plt.ylabel("score")
plt.xlabel("learning rate")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22a9a9d90>
```



모델 설명력에 대한 여러 그래프의 확인 내용. y_label 은 'score'로 동일하지만, x_label은 'n_estimators' , 'min samples leaf', 'depth', 'learning rate'로 네 가지의 다른 그래프들을 확인할 수 있다.

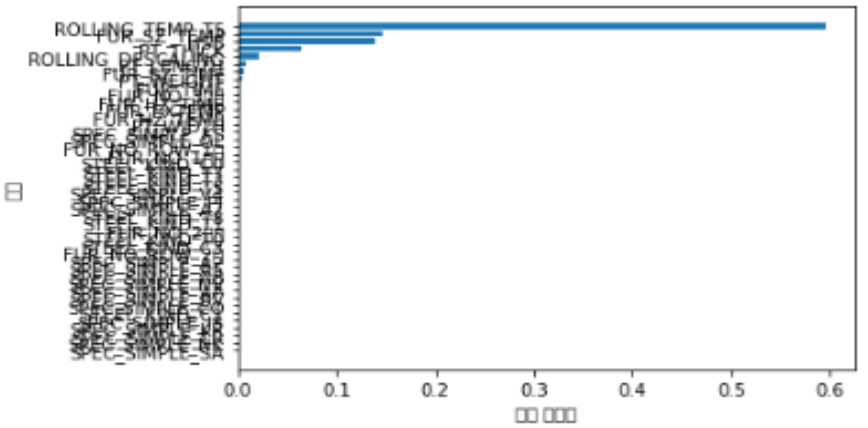
Gradient Boosting 설명변수 중요도

```
# 변수명 저장
v_feature_name = df_train_x.columns

# tree.feature_importances_로 설명변수 중요도 확인 및 테이블로 저장
df_importance = pd.DataFrame()
df_importance['Feature'] = v_feature_name
df_importance['Importance'] = gb_final.feature_importances_

# df_feature_importance의 테이블을 중요도별로 정렬
df_importance.sort_values('Importance', ascending=False, inplace=True)
df_importance.round(3)
```

	Feature	Importance
10	ROLLING_TEMP_T5	0.596
6	FUR_SZ_TEMP	0.146
11	HSB	0.138
0	PT_THICK	0.065
12	ROLLING_DESCALING	0.022
2	PT_LENGTH	0.008

'ROLLING_TEMP_T5' 의 Importance는 0.596, **'FUR_SZ_TEMP'** 의 Importance는 0.146,
'HSB'의 Importance는 0.138로 세 가지 설명변수가 SCALE에 가장 영향을 많이 주는 변수임을 알 수 있다

```
#설명변수 중요도 그래프
#중요도가 높은 변수를 상위에 그림.
df_importance.sort_values("Importance", ascending=True, inplace=True)
coordinates = range(len(df_importance))
plt.barh(y=coordinates, width = df_importance["Importance"])
plt.yticks(coordinates, df_importance["Feature"])
plt.xlabel("변수 중요도")
plt.ylabel("변수")
```



압연온도(ROLLING_TEMP_T5). 가열로 균열대 시간(FUR_SZ_TEMP), 고열의 스케일 브레이커(HSB) 순으로 SCALE에 영향이 크다고 해석할 수 있음

로지스틱 회귀분석

```
log_model=Logit.from_formula("""SCALE ~ PT_THICK+PT_WIDTH+PT_LENGTH+PT_WEIGHT+FUR_HZ_TEMP+FUR_HZ_TIME+FUR_SZ_TEMP+FUR_SZ_TIME+FUR_TIME+FUR_EXTEMP+ROLLING_TEMP_T5+HSB+ROLLING_DESCALING""", df_train)
log_result = log_model.fit()
print(log_result.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.250279
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:                  SCALE   No. Observations:                  426
Model:                          Logit   Df Residuals:                      412
Method:                           MLE   Df Model:                           13
Date:                Wed, 17 Aug 2022   Pseudo R-squ.:                  0.6004
Time:                        09:11:22   Log-Likelihood:                -106.62
converged:                      False   LL-Null:                       -266.80
Covariance Type:            nonrobust   LLR p-value:                 1.299e-60
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -122.9450   7.55e+05     -0.000      1.000   -1.48e+06    1.48e+06
PT_THICK      -0.0005      0.049     -0.011      0.991      -0.097       0.096
PT_WIDTH      -0.0018      0.001     -3.345      0.001      -0.003      -0.001
PT_LENGTH  -1.197e-05   3.22e-05     -0.372      0.710   -7.51e-05    5.12e-05
PT_WEIGHT  -3.756e-06   7.98e-06     -0.471      0.638   -1.94e-05    1.19e-05
FUR_HZ_TEMP   0.0060      0.018      0.328      0.743      -0.030       0.042
FUR_HZ_TIME   0.0178      0.011      1.628      0.104      -0.004       0.039
FUR_SZ_TEMP   0.0693      0.070      0.992      0.321      -0.068       0.206
FUR_SZ_TIME  -0.0133      0.017     -0.794      0.427      -0.046       0.020
FUR_TIME     -0.0094      0.006     -1.502      0.133      -0.022       0.003
FUR_EXTEMP    0.0411      0.058      0.710      0.478      -0.072       0.155
ROLLING_TEMP_T5   0.0370      0.006      5.730      0.000       0.024       0.050
HSB         -33.8806   7.55e+05  -4.49e-05      1.000   -1.48e+06    1.48e+06
ROLLING_DESCALING   -0.7709      0.254     -3.031      0.002      -1.269      -0.272
==============================================================================
```

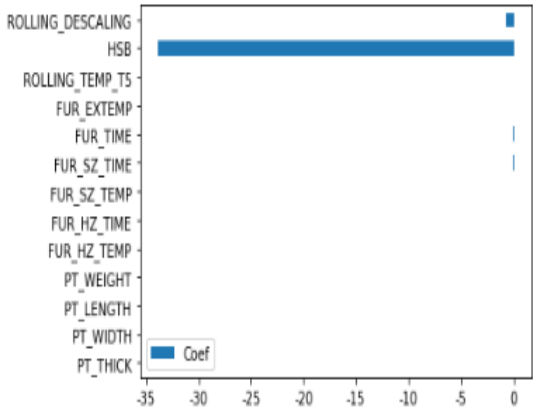
로지스틱 회귀분석을 하기 위하여 SCALE 을 비롯한 다른 변수들을 입력한다

로지스틱 회귀분석

```
Train 예측/분류 결과
Accuracy: 0.899

Confusion Matrix:
 [[272  18]
 [ 25 111]]

              precision    recall  f1-score   support

           0      0.916     0.938     0.927       290
           1      0.860     0.816     0.838       136

    accuracy                          0.899       426
   macro avg      0.888     0.877     0.882       426
weighted avg      0.898     0.899     0.898       426

test 예측/분류 결과
Accuracy: 0.814

Confusion Matrix:
 [[105  16]
 [ 18  44]]

              precision    recall  f1-score   support

           0      0.854     0.868     0.861       121
           1      0.733     0.710     0.721        62

    accuracy                          0.814       183
   macro avg      0.793     0.789     0.791       183
weighted avg      0.813     0.814     0.813       183
```

1)모델의 Test 데이터 정분류율은 81.4 %
2) 분류 내용 -> 실제 0을 0으로 분류 105건 -> 실제 1을 0으로 분류 18 건 ....
3) "1" 기준 f1 score는 72.1% (Precision : 73.3%, Recall : 71.0%)

```
df_logistic_coef = pd.DataFrame({'Coef': log_result.params.values[1:]}, index = log_model.exog_names[1:])
df_logistic_coef.plot.barh(y='Coef');
```



'Rolling_Descaling', 'HSB'이 높아질수록 SCALING 이 발생할 가능성이 오히려 낮아짐

KNN

```
# 데이터 구성 : Series, DataFrame
import pandas as pd
# 데이터 시각화
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
```

```
# 데이터 구성하기 - 데이터 분할
# 설명변수, 목표변수 데이터 구분
df_raw_x = df_raw_dummy.drop('SCALE', axis=1 ,inplace=False)
df_raw_y = df_raw_dummy['SCALE']

df_train_x, df_test_x, df_train_y, df_test_y = train_test_split(
    df_raw_x, df_raw_y, test_size=0.3, random_state=1234)

print('분할 전 설명변수 데이터:', df_raw_x.shape)
print('분할 후 설명변수 데이터:Train', df_train_x.shape, 'Test',df_test_x.shape)
```

```
분할 전 설명변수 데이터: (609, 44)
분할 후 설명변수 데이터:Train (426, 44) Test (183, 44)
```

```
knn_uncusotomized = KNeighborsClassifier()
knn_uncusotomized.fit(df_train_x,df_train_y)
print("Accuracy on training set : {:.3f}".format(knn_uncusotomized.score(df_train_x,df_train_y)))
print("Accuracy on test set : {:.3f}".format(knn_uncusotomized.score(df_test_x,df_test_y)))
```

```
Accuracy on training set : 0.822
Accuracy on test set : 0.689
```

KNN을 진행하기 위해서 데이터 구성 및 데이터 시각화, KNeighborsClassifier를 호출

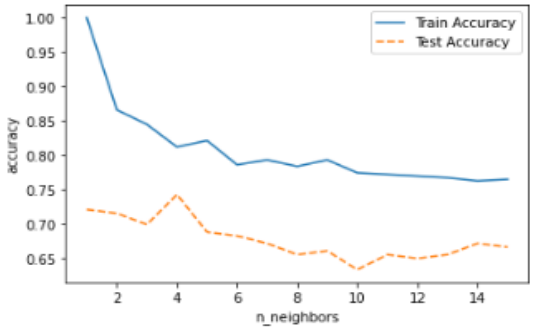
데이터 구성하기 및 데이터 분할, knn_uncustomized를 통하여 Accuracy on training set : 0.822, Accuracy on test set : 0.689 라는 값을 얻을 수 있었다.

모델링 & 요약

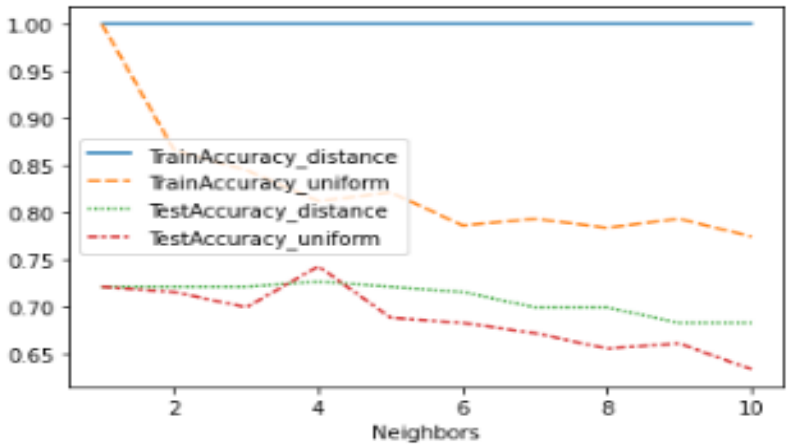
KNN

```
plt.plot(para_n_neighbors, train_accuracy, linestyle="-", label = "Train Accuracy")
plt.plot(para_n_neighbors, test_accuracy, linestyle="--", label = "Test Accuracy")
plt.ylabel("accuracy"); plt.xlabel("n_neighbors")

plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22a741b10>
```

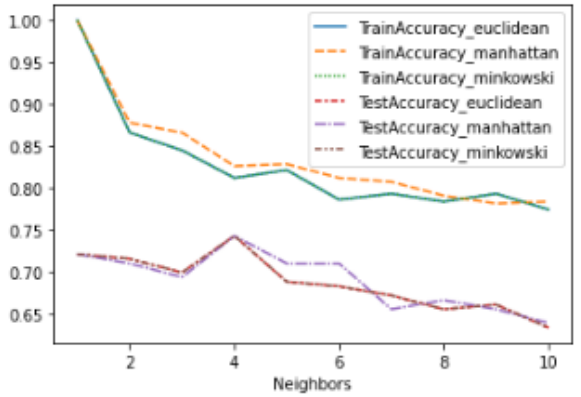



```
sns.lineplot(data=df_accuracy_weights_pivot)

weights="uniform"
```




```
df_accuracy_metric_pivot = df_accuracy_metric.pivot(index = "Neighbors", columns="Metric",
values = ["TrainAccuracy", "TestAccuracy"])
level0 = df_accuracy_metric_pivot.columns.get_level_values(0)
level1 = df_accuracy_metric_pivot.columns.get_level_values(1)
df_accuracy_metric_pivot.columns = level0 + "_" + level1
sns.lineplot(data=df_accuracy_metric_pivot)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa22a504ed0>
```



모델 설명력에 대한 여러 그래프의 확인 내용.
Train Accuracy, Test Accuracy에 관한 그래프,
weights_pivot에 따른 그래프, Euclidean, Manhattan,
minkowski 에 따른 그래프 등을 확인할 수 있다.

KNN

```
knn_model = KNeighborsClassifier(n_neighbors = 4, weights = 'uniform',metric = 'euclidean')
knn_model.fit(df_train_x, df_train_y)
y_pred = knn_model.predict(df_test_x)

print('train data accuracy : {0:.3f}'.format(knn_model.score(df_train_x, df_train_y)))

print('test data accuracy : {0:.3f}'.format(knn_model.score(df_test_x, df_test_y)))

print('Confusion matrix : ₩n{}'.format(confusion_matrix(df_test_y,y_pred)))

print(classification_report(df_test_y, y_pred,  digits = 3))
```

```
train data accuracy : 0.812
test data accuracy : 0.743
Confusion matrix :
[[109  12]
 [ 35  27]]
              precision    recall  f1-score   support

           0      0.757     0.901     0.823       121
           1      0.692     0.435     0.535        62

    accuracy                          0.743       183
   macro avg      0.725     0.668     0.679       183
weighted avg      0.735     0.743     0.725       183
```

KNN 분석 결과, 'train data accuracy' = 0.812, 'test data accuracy' = 0.743 의 값을 얻을 수 있다.

SVM

```
# 데이터 구성 : Series, DataFrame
import pandas as pd
import numpy as np
# 데이터 시각화, 한글폰트
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rc('font', family = 'Malgun Gothic')

# scaling
from sklearn.preprocessing import StandardScaler
# data split
from sklearn.model_selection import train_test_split
# SVM
from sklearn.svm import SVC
# 최적 모델, 파라미터 탐색
from sklearn.model_selection import GridSearchCV

# 분류모델 평가 함수
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
```

```
# 데이터 구성하기 - 데이터 분할
# 설명변수, 목표변수 데이터 구분
df_raw_x = df_raw_dummy.drop('SCALE', axis=1 ,inplace=False)
df_raw_y = df_raw_dummy['SCALE']

df_train_x, df_test_x, df_train_y, df_test_y = train_test_split(
    df_raw_x, df_raw_y, test_size=0.3, random_state=1234)

print('분할 전 설명변수 데이터:', df_raw_x.shape)
print('분할 후 설명변수 데이터:Train', df_train_x.shape, 'Test',df_test_x.shape)
```

```
분할 전 설명변수 데이터: (609, 44)
분할 후 설명변수 데이터:Train (426, 44) Test (183, 44)
```

```
# SVC 모델 생성 - 기본 옵션으로 모델 생성
svm_uncustomized = SVC(random_state = 1234)
svm_uncustomized.fit(df_train_x, df_train_y)
# train 데이터 셋 정확도
print('Accuracy on training set: {:.3f}'.format(svm_uncustomized.score(df_train_x, df_train_y)))
# test데이터 정확도
print('Accuracy on test set: {:.3f}'.format(svm_uncustomized.score(df_test_x, df_test_y)))
```

```
Accuracy on training set: 0.681
Accuracy on test set: 0.661
```

SVM을 진행하기 위해서 데이터 구성 및 데이터 시각화, SVC를 호출

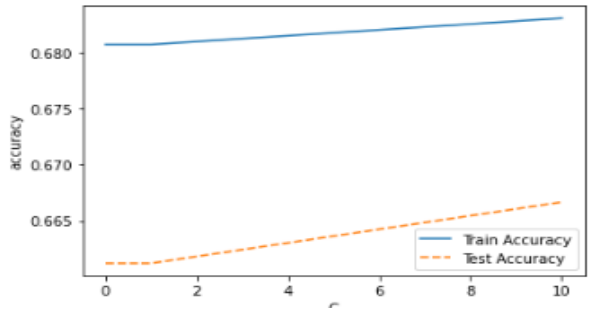
데이터 구성하기 및 데이터 분할, svm_uncustomized를 통하여 Accuracy on training set : 0.681, Accuracy on test set : 0.661 라는 값을 얻을 수 있었다.

모델링 & 요약

SVM

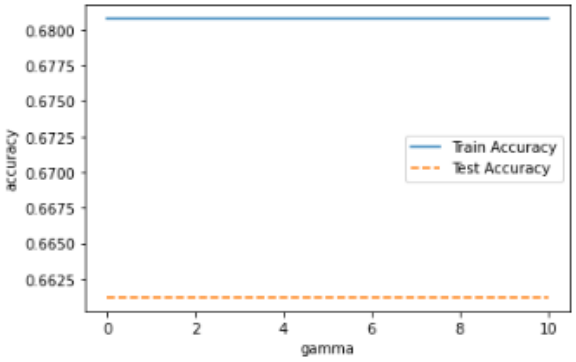
```
# 모델 정확도 그래프 확인
plt.plot(para_c, train_accuracy, linestyle = '-', label = 'Train Accuracy')
plt.plot(para_c, test_accuracy, linestyle = '--', label = 'Test Accuracy')
plt.ylabel('accuracy'); plt.xlabel('C')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22a436790>
```



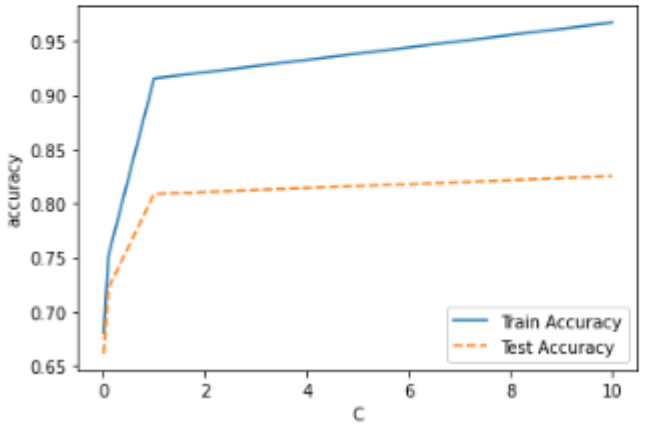

```
# 모델 정확도 그래프 확인
plt.plot(para_gamma, train_accuracy, linestyle = '-', label = 'Train Accuracy')
plt.plot(para_gamma, test_accuracy, linestyle = '--', label = 'Test Accuracy')
plt.ylabel('accuracy'); plt.xlabel('gamma')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22a3b81d0>
```




```
# 모델 정확도 그래프 확인
plt.plot(para_c, train_accuracy, linestyle = '-', label = 'Train Accuracy')
plt.plot(para_c, test_accuracy, linestyle = '--', label = 'Test Accuracy')
plt.ylabel('accuracy'); plt.xlabel('C')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa22a3a0250>
```



모델 정확도에 대한 여러 그래프의 확인 내용.
Train Accuracy, Test Accuracy에 관한 그래프, accuracy와 gamma 에 따른 그래프 등을 확인할 수 있다.

SVM

```
# 결론 도출 - 최종 모델 선택

svc_final = SVC(gamma = 0.1, C = 1, random_state= 1234)
svc_final.fit(df_scaled_train_x, df_train_y)
y_pred = svc_final.predict(df_scaled_test_x)
# train 데이터 셋 정확도
print("Accuracy on training set: {:.3f}".format(svc_final.score(df_scaled_train_x, df_train_y)))
# test 데이터 셋 정확도
print("Accuracy on test set: {:.3f}".format(svc_final.score(df_scaled_test_x, df_test_y)))
# confusion matrix
print("Confusion matrix: ₩n{}".format(confusion_matrix(df_test_y, y_pred)))
# 목표변수의 빈도 불균형 : f1 score로 모델 평가
print(classification_report(df_test_y, y_pred, digits=3))
```

```
Accuracy on training set: 0.958
Accuracy on test set: 0.770
Confusion matrix:
[[110  11]
 [ 31  31]]
              precision    recall  f1-score   support

           0      0.780     0.909     0.840       121
           1      0.738     0.500     0.596        62

    accuracy                          0.770       183
   macro avg      0.759     0.705     0.718       183
weighted avg      0.766     0.770     0.757       183
```

SVM 분석 결과, 'train data accuracy' = 0.958, 'test data accuracy' = 0.770 의 값을 얻을 수 있다.

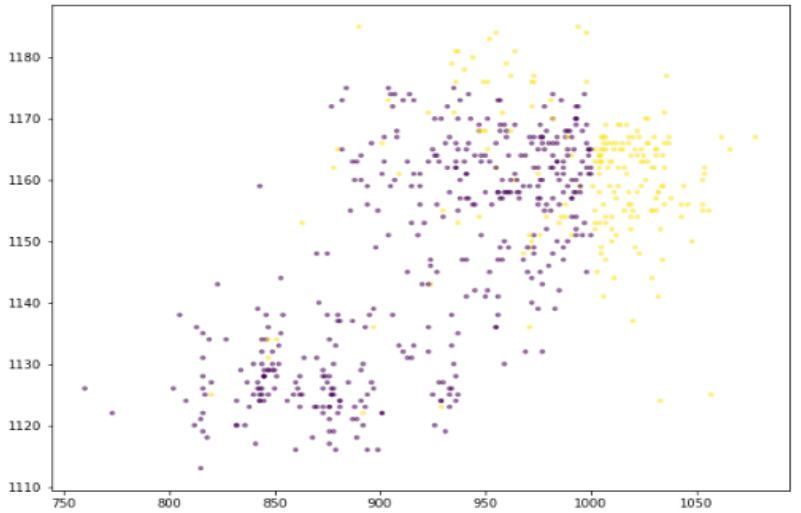
의사결정 트리

```
# 데이터 구성하기 - 데이터 분할
# 설명변수, 목표변수 데이터 구분
df_raw_x = df_raw_dummy.drop('SCALE', axis=1 ,inplace=False)
df_raw_y = df_raw_dummy['SCALE']

df_train_x, df_test_x, df_train_y, df_test_y = train_test_split(
    df_raw_x, df_raw_y, test_size=0.3, random_state=1234)

print('분할 전 설명변수 데이터:', df_raw_x.shape)
print('분할 후 설명변수 데이터:Train', df_train_x.shape, 'Test',df_test_x.shape)
```

```
분할 전 설명변수 데이터: (609, 44)
분할 후 설명변수 데이터:Train (426, 44) Test (183, 44)
```

```
# Over-Sampling : SMOTE

# 목표변수 빈도 확인
print(df_raw.value_counts(['SCALE']),'₩n')
print('SCALE=1 비율', df_raw.value_counts(df_raw['SCALE']==1)/len(df_raw))

# 목표변수 산점도 확인
plt.figure(figsize=(10,8))
plt.scatter(df_raw['ROLLING_TEMP_T5'],df_raw['FUR_SZ_TEMP'], c=df_raw['SCALE'],s=10,alpha=0.5)
plt.show()
```

```
SCALE
0        411
1        198
dtype: int64

SCALE=1 비율 SCALE
False    0.674877
True     0.325123
dtype: float64
```

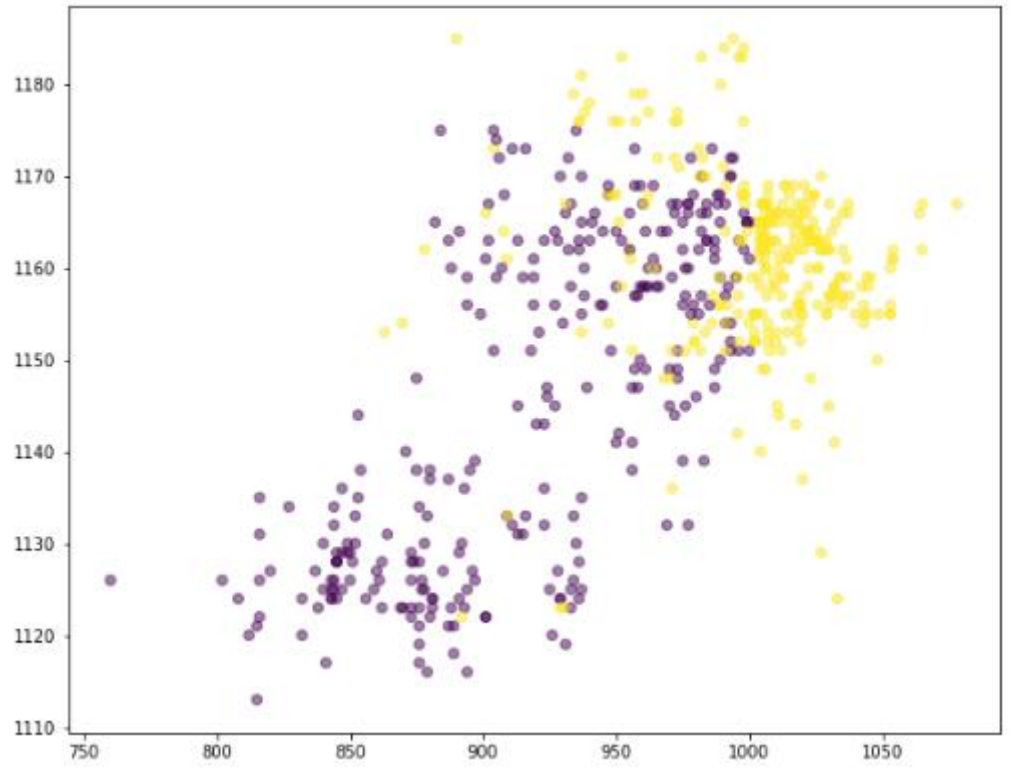


데이터 구성하기 및 분할 전 설명변수 데이터, 분할 후 설명변수 데이터를 확인할 수 있다. 또한 , Over-Sampling : SMOTE를 통하여 얻을 수있는 목표변수 산점도를 확인할 수 있다.

의사결정 트리

```
# 데이터 결합
df_resampled = pd.concat([x_resampled, y_resampled], axis=1)
print(df_resampled.head())
# 목표변수 산점도 확인
plt.figure(figsize=(10,8))
plt.scatter(df_resampled['ROLLING_TEMP_T5'], df_resampled['FUR_SZ_TEMP'],
            c=df_resampled['SCALE'], alpha=0.5)
plt.show()
```

[5 rows x 45 columns]




```
plt.show()
```

0	30.23	1940.0	34797	16020	1119	130.0	
1	14.09	3284.0	51234	93050	1142	75.0	
2	44.39	2040.0	27501	39100	1113	124.0	
3	18.10	3094.0	41786	91850	1159	132.0	
4	18.10	3094.0	41786	91850	1146	108.0	

	FUR_SZ_TEMP	FUR_SZ_TIME	FUR_TIME	FUR_EXTEMP	...	SPEC_SIMPLE_JS	#
0	1120	65.0	324	1112	...	0	
1	1131	122.0	353	1125	...	1	
2	1120	82.0	334	1113	...	0	
3	1164	54.0	358	1155	...	1	
4	1163	88.0	351	1148	...	1	

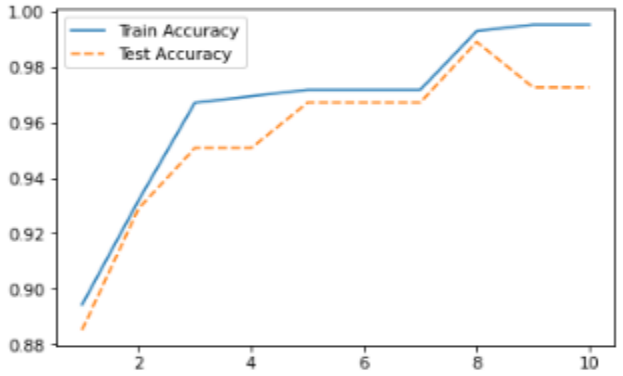
	SPEC_SIMPLE_KR	SPEC_SIMPLE_KS	SPEC_SIMPLE_LR	SPEC_SIMPLE_NK	#
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	

	SPEC_SIMPLE_NV	SPEC_SIMPLE_PI	SPEC_SIMPLE_SA	SPEC_SIMPLE_V4	SCALE
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

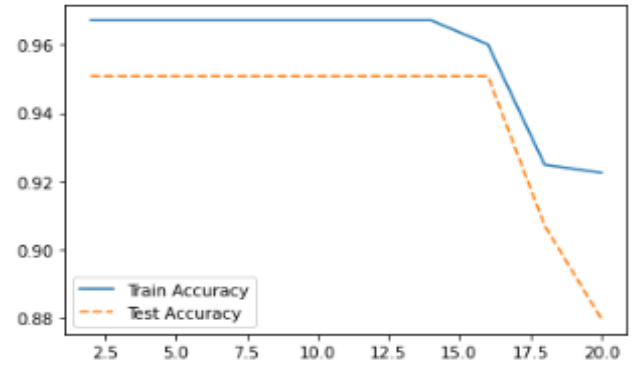
데이터 결합 이후 목표변수 산점도를 확인할 수 있다

의사결정 트리

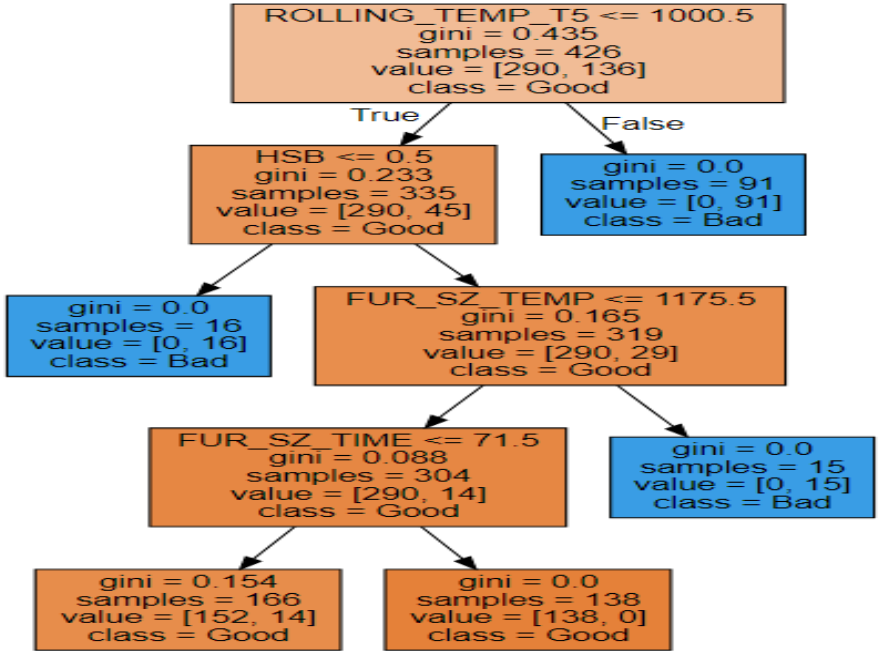
```
# 모델 정확도에 대한 그래프 확인
plt.plot(para_depth, train_accuracy, linestyle='-', label='Train Accuracy')
plt.plot(para_depth, test_accuracy, linestyle='--', label='Test Accuracy')
plt.legend();
```




```
# 모델 정확도 그래프 확인
plt.plot(para_leaf, train_accuracy, linestyle='-', label='Train Accuracy')
plt.plot(para_leaf, test_accuracy, linestyle='--', label='Test Accuracy')
plt.legend();
```




```
# 최종 모델 시각화
# tree_final.dot으로 결과 저장
export_graphviz(tree_final, out_file='tree_final.dot', class_names = ['Good', 'Bad'],
                feature_names = v_feature_name,impurity=True,filled=True)
# tree_final.dot 그리기
with open('tree_final.dot') as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```



모델 정확도 그래프 및 최종 모델 시각화를 확인

의사결정 트리

```
# 평가
y_pred =tree_final.predict(df_test_x)
print('Accuracy: {0:.3f}₩n'.format(tree_final.score(df_test_x, df_test_y)))
print('Confusion matrix: ₩n{}'.format(confusion_matrix(df_test_y, df_test_y)))
# 목표변수의 빈도 불균형 : f1 score로 모델 평가
print(classification_report(df_test_y, y_pred, digits=3))
```

```
Accuracy: 0.951

Confusion matrix:
[[121   0]
 [  0  62]]
              precision    recall  f1-score   support

           0      0.931     1.000     0.964       121
           1      1.000     0.855     0.922        62

    accuracy                          0.951       183
   macro avg      0.965     0.927     0.943       183
weighted avg      0.954     0.951     0.950       183
```

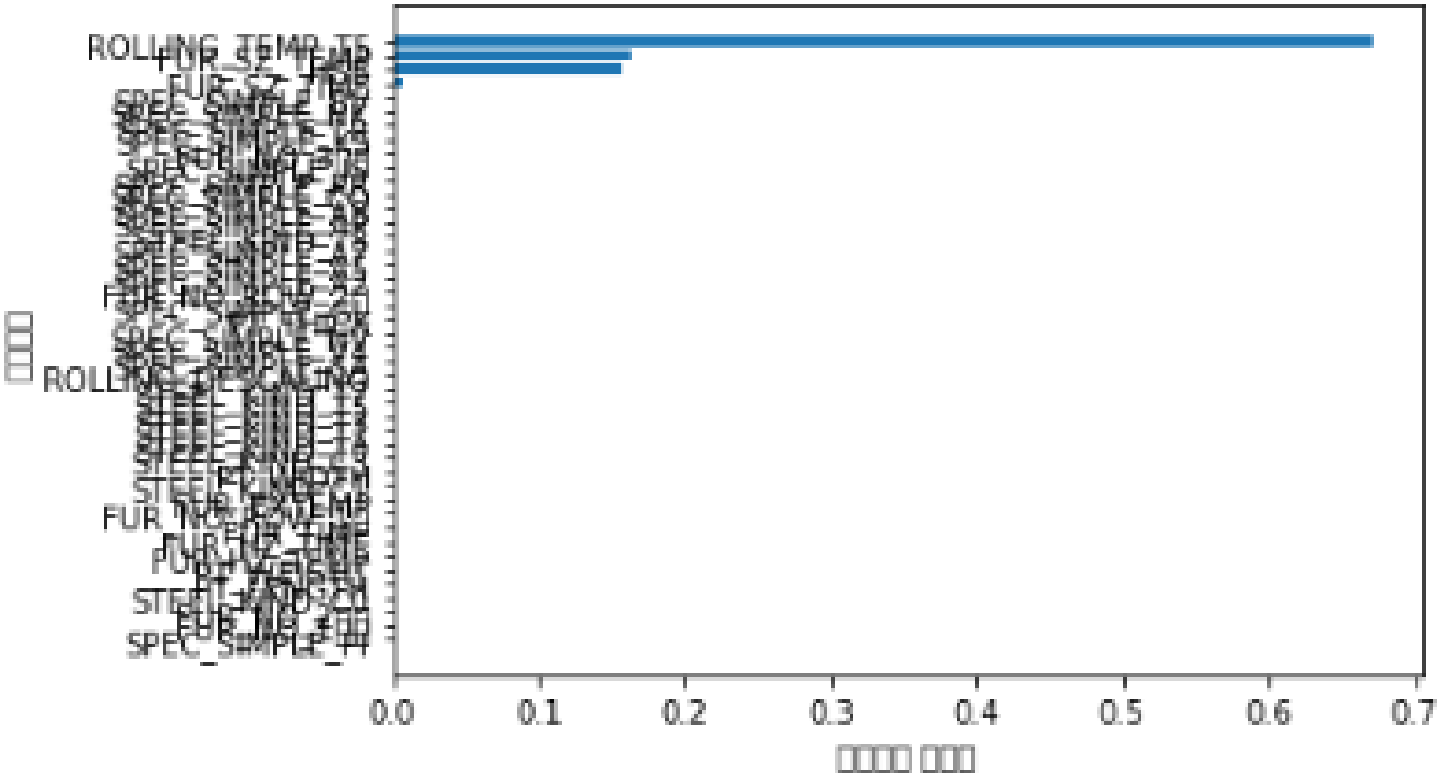
Accuracy, f1-score 등을 알 수 있다

```
# tree.feature_importance_ 로 설명변수 중요도 확인 및 테이블로 저장
df_importance = pd.DataFrame()
df_importance['Feature'] = v_feature_name
df_importance['Importance'] = tree_final.feature_importances_

# df_feature_importance의 테이블을 중요도 순으로 정렬
df_importance.sort_values('Importance',ascending=False, inplace=True)
df_importance.round(3)
```

	Feature	Importance
10	ROLLING_TEMP_T5	0.672
6	FUR_SZ_TEMP	0.163
11	HSB	0.158
7	FUR_SZ_TIME	0.007

Feature_importance 로 설명변수 중요도 확인 및 테이블을 중요도 순으로 정렬한다.

의사결정 트리



중요 설명변수 : 압연온도(ROLLING_TEMP_T5). 가열로 균열대 시간(FUR_SZ_TEMP), 고열의 스케일 브레이커(HSB) 순으로 영향이 크다고 해석할 수 있음

결론 및 대안제시

BEFORE

After

데이터 기반 정책

Normalizer 적용을 통한 Heatmap 출력, Random Forest, Gradient Boosting, 로지스틱 회귀분석, KNN, SVM, 의사결정 트리를 통하여, **압연온도(ROLLING_TEMP_T5). 가열로 균열대 시간(FUR_SZ_TEMP), 고열의 스케일 브레이커(HSB)** 조절을 통해 Scale 불량을 줄일 수 있을 것이다