STEEL_DATA를 활용한 불량률 예측 분석

1조

김성민, 염동화, 이규진, 이주상

목차 a table of contents

Part 1 # 분석배경

Steel Mill, Steelworks

'산업의 쌀' 생산하는 제철소 !

1. 철강은 인류의 가장 큰 산업

2 . 산업의 기초는 철강

3 . 철강의 다양한 활용은 고급 및 핵심기술

Part 1 분석배경

철강 '불량 제로' 만든건 데이터 전문가...'디지털 협업'이 R&D 완성

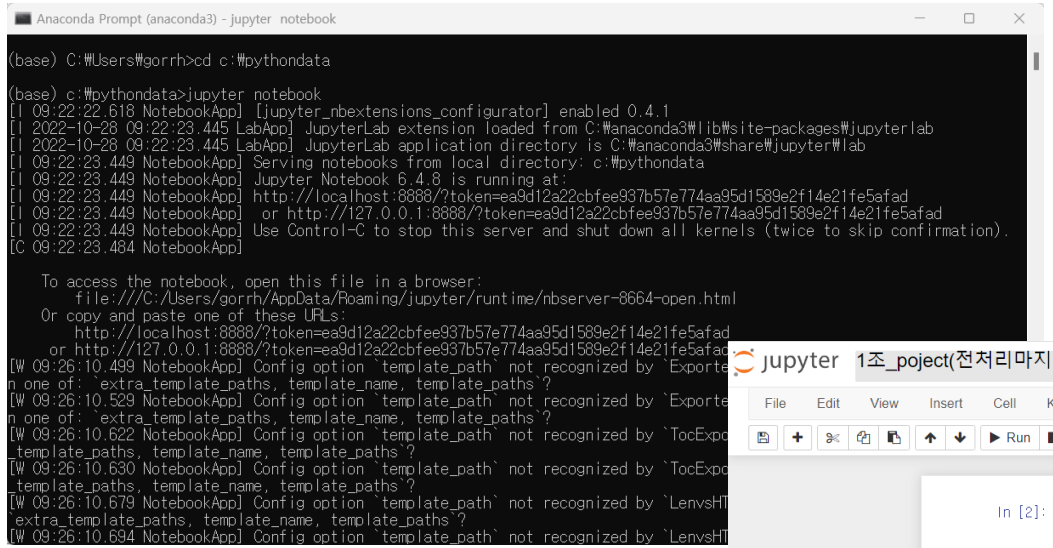
獨 철강설비 기업의 도전
핵심은 디지털·데이터 공유

"앞으로 모든 회사는 인공지능(AI) 기업이 될 것이다. AI 기반 기술 플랫폼 구축이 경쟁 우위의 기본이다. 시장 기회를 빠르게 포착하고, 위기에 신속히 대응하는 것은 기술 플랫폼에 달렸다."

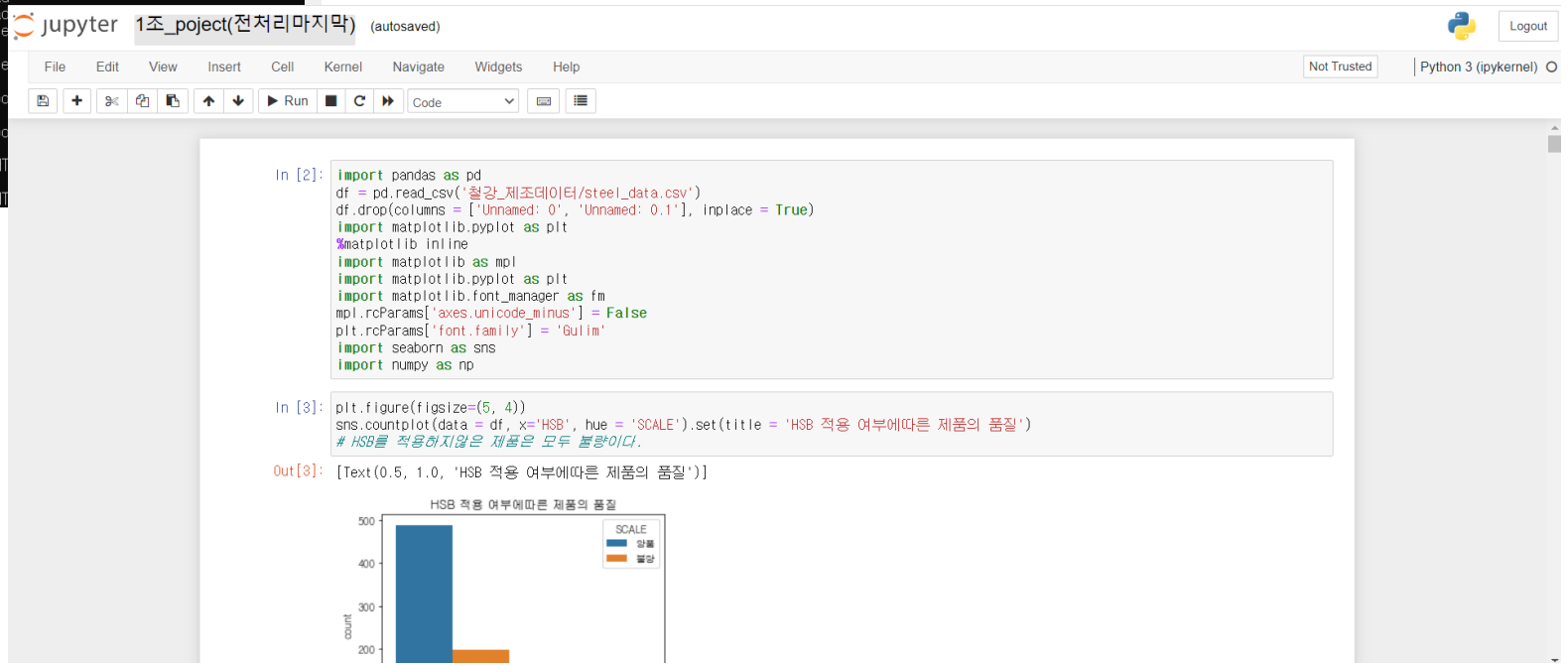
독일의 철강설비 엔지니어링 기업 SMS는 올초 '데이터 챌린지'를 열었다. 인공지능(AI)을 활용한 머신러닝을 철강 공정에 도입해 생산량을 늘릴 방안을 찾기 위한 행사였다. 독일 명문대에서 데이터 과학을 전공하는 학부생과 대학원생 25명이 참여했다. 철강업에 대한 지식이 전무한 데이터 전문가들이었다.

이들이 내놓은 결과물은 놀라웠다. 철강업계 전문가들이 수십 년간 해결하지 못한 난제를 두 달 만에 풀어냈다. 철강 주조 과정에서 불량품을 빠르게 잡아내는 알고리즘을 개발한 것이다. 미국의 한 철강회사 공정에서 나온 3만여 개 데이터를 활용해 불량품을 잡아내는 정확도를 100%까지 높였다. 철강업계에선 현장에 적용하면 비용 감소, 탄소 절감 등의 효과로 이어질 '혁신적 성과'라는 평가나 나왔다. SMS는 "젊은 데이터 과학자들의 혁신적 접근은 우리에게 엄청난 도움이 될 것"이라고 밝혔다.

Part 1 분석배경



Anaconda 3
Jupyter notebook



Part 2 데이터 분석

칼럼 명	칼럼 설명	칼럼 명	칼럼 설명
PLATE_NO	플랜트 고유 ID	FUR_HZ_TEMP	HZ가열로 온도(℃)
ROLLING_DATE	제작일자	FUR_HZ_TIME	HZ가열로 시간(초)
SCALE	양품/불량	FUR_SZ_TEMP	SZ가열로 온도(℃)
SPEC	품명	FUR_SZ_TIME	SZ가열로 시간(초)
STEEL_KIND	종류	FUR_TIME	가열로 내부에 있었던 시간(초)
PT_THK	두께(mm)	FUR_EXTEMP	가열로 추출온도(℃)
PT_WDTH	너비(mm)	ROLLING_TEMP_T5	롤링 온도(℃)
PT_LTH	길이(mm)	HSB	HSB 적용여부
PT_WGT	무게(kg)	ROLLING_DESCALING	ROLLING DESCALING 작업 횟수
FUR_NO	가열로 NO	WORK_GR	작업그룹
FUR_NO_ROW	가열로 ROW		

PLATE_NO	ROLLING_DATE	SCALE	SPEC	STEEL_KIND	PT_THK	PT_WDTH	PT_LTH	PT_WGT	FUR_NO	...
PB562774	2008-08-01 00:00:15	양품	AB/EH32-TM	T1	32.25	3707	15109	14180	1호기	...
PB562775	2008-08-01 00:00:16	양품	AB/EH32-TM	T1	32.25	3707	15109	14180	1호기	...
PB562776	2008-08-01 00:00:59	양품	NV-E36-TM	T8	33.27	3619	19181	18130	2호기	...
PB562777	2008-08-01 00:01:24	양품	NV-E36-TM	T8	33.27	3619	19181	18130	2호기	...
PB562778	2008-08-01 00:01:44	양품	BV-EH36-TM	T8	38.33	3098	13334	12430	3호기	...

FUR_SZ_TIME	FUR_TIME	FUR_EXTEMP	ROLLING_TEMP_T5	HSB	ROLLING_DESCALING	WORK_GR
59	282	1133	934	적용	8	2조
53	283	1135	937	적용	8	2조
55	282	1121	889	적용	8	3조
68	316	1127	885	적용	8	3조
48	314	1128	873	적용	8	1조

SCALE	count
불량	231
양품	489

불량발생률
약 32%

Part 2 데이터 분석

1 작업 시간

STEP 1		STEP 2		STEP 3
시간 별 양품 및 불량 개수 측정	>>	불량률 계산을 위한 데이터 전처리	>>	시간 별 불량률 측정

Part 2 데이터 분석

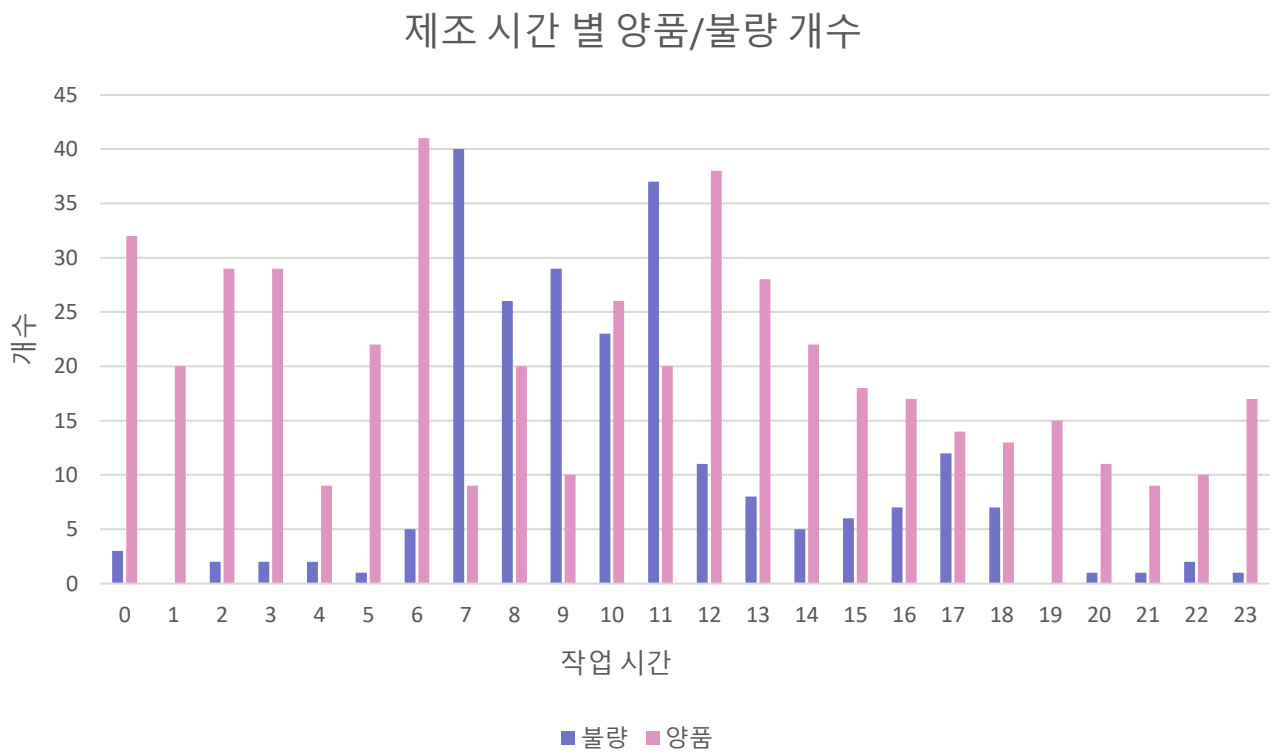
- 작업 시간 데이터

```
df['ROLLING_DATE']
```

```
0     2008-08-01 00:00:15
1     2008-08-01 00:00:16
2     2008-08-01 00:00:59
3     2008-08-01 00:01:24
4     2008-08-01 00:01:44
              ...
715   2008-08-02 13:35:36
716   2008-08-02 13:35:02
717   2008-08-02 14:40:00
718   2008-08-02 13:35:19
719   2008-08-02 14:40:53
Name: ROLLING_DATE, Length: 720, dtype: datetime64[ns]
```

```
df['rolling_hour'] = df['ROLLING_DATE'].dt.hour
df['rolling_hour']
# ROLLING_DATE 를 시간만 추출
```

```
0       0
1       0
2       0
3       0
4       0
       ..
715    13
716    13
717    14
718    13
719    14
Name: rolling_hour, Length: 720, dtype: int64
```



Part 2 데이터 분석

- 작업 시간 데이터

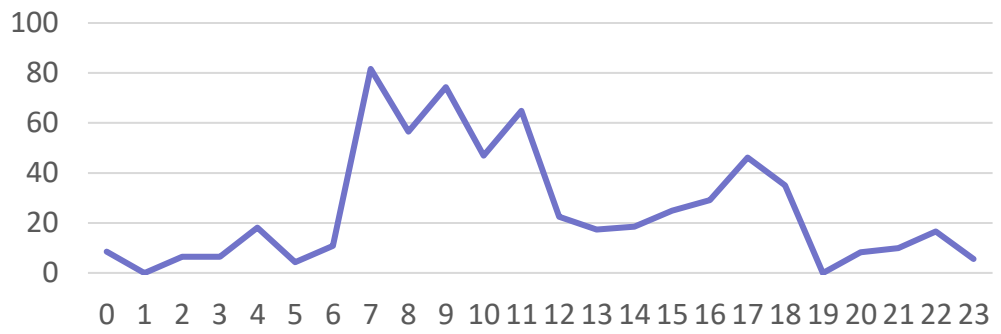
```
def convert(SCALE):
    if SCALE == "불량":
        return 1
    else:
        return 0
df['SCALE_no'] = df['SCALE'].apply(lambda x: convert(x))
```

양품을 0, 불량을 1 로 지정한 열 생성

ROLLING_TEMP_T5	HSB	ROLLING_DESCALING	WORK_GR	SCALE_no
934	적용	8	2조	0
937	적용	8	2조	0
889	적용	8	3조	0
885	적용	8	3조	0
873	적용	8	1조	0

```
df_scale_hour = df.groupby(
    ['rolling_hour'], as_index = False)[['SCALE_no']].agg("mean")
df_scale_hour.head()
```

	rolling_hour	SCALE_no
0	0	0.085714
1	1	0.000000
2	2	0.064516
3	3	0.064516
4	4	0.181818



Part 2 데이터 분석

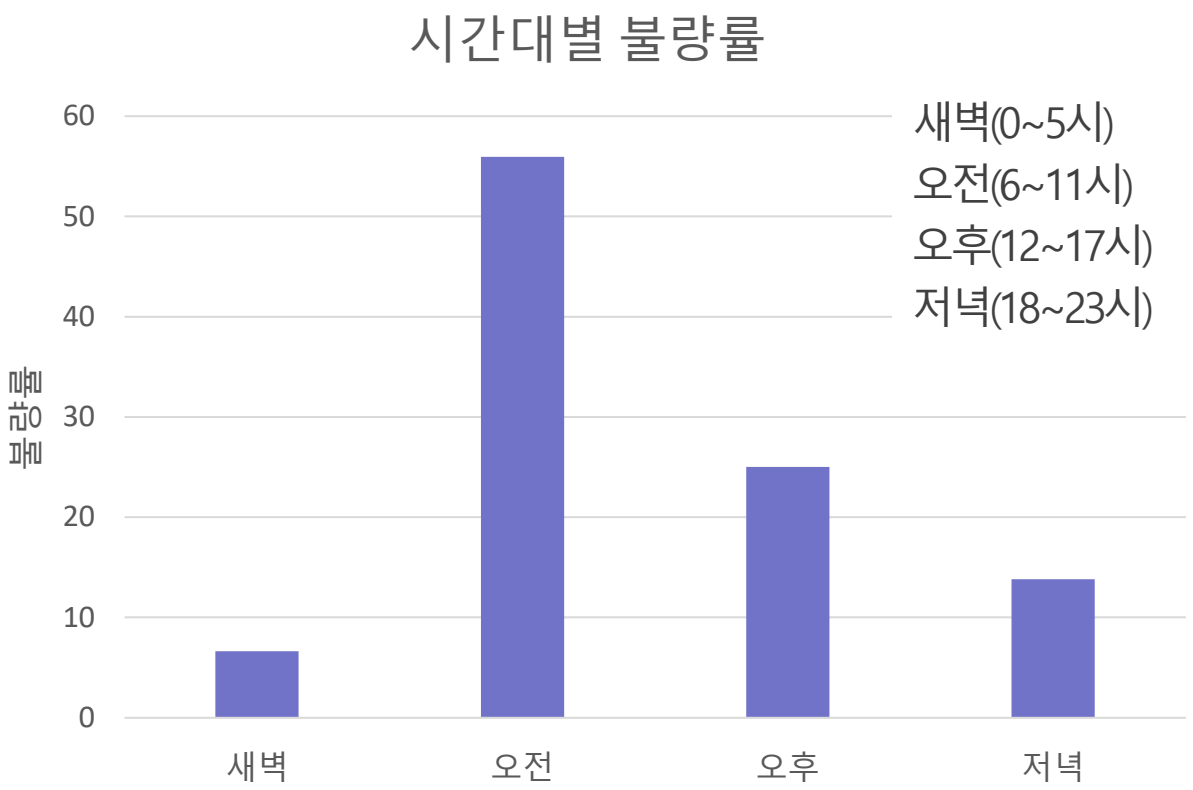
- 작업 시간 데이터

```
def slot(x):
    if 0<=x<=5:
        return "새벽"
    elif 6<=x<=11:
        return "오전"
    elif 12<=x<=17:
        return "오후"
    else :
        return "저녁"
df['rolling_slot'] = df['rolling_hour'].apply(slot)
df['rolling_slot']
# ROLLING_DATE를 시간만 추출해 시간대로 분류 0~5시를 새벽 6~11시를
# 오전 12~17시를 오후 18~23시를 저녁으로 지정
```

```
0      새벽
1      새벽
2      새벽
3      새벽
4      새벽
      ..
715    오후
716    오후
717    오후
718    오후
719    오후
Name: rolling_slot, Length: 720, dtype: object
```

```
df_rolling_slot = df.groupby(
    ['rolling_slot'], as_index = False)[['SCALE_no']].agg("mean")
df_rolling_slot
# 시간대별로 불량률을 계산
```

	rolling_slot	SCALE_no
0	새벽	0.066225
1	오전	0.559441
2	오후	0.250000
3	저녁	0.137931



오전 시간대에 가장 높은 불량률

2 Steel 종류

STEP 1

제품 별 상대 생산율

>>

STEP 2

제품 별 상대 불량률

STEP 3

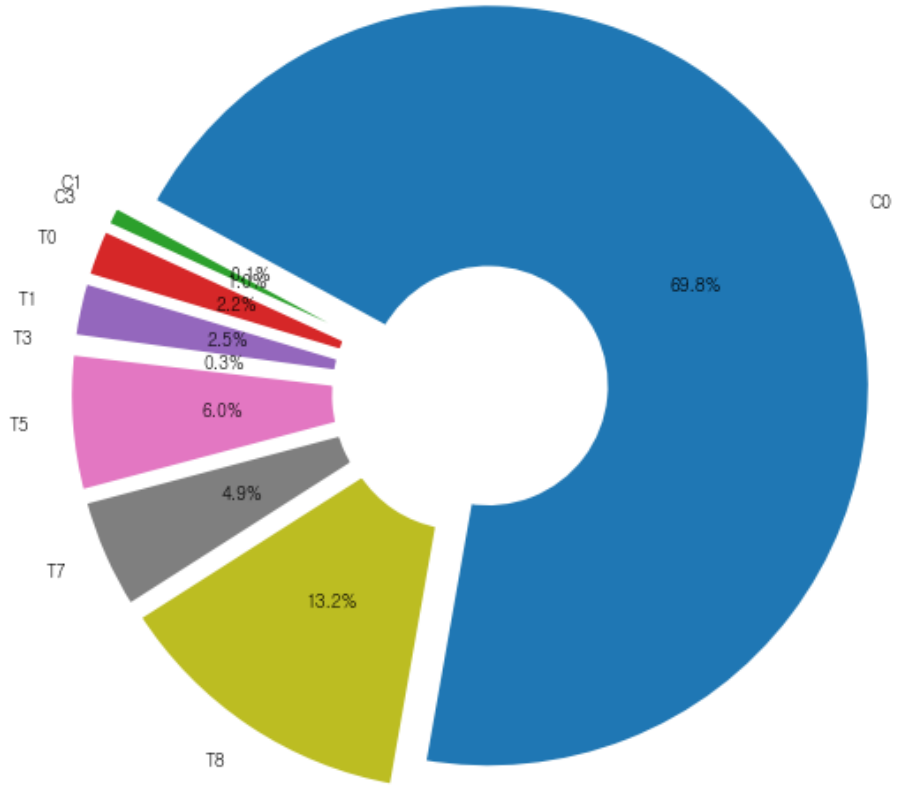
Part 2 # 데이터 분석

- 철강 종류

STEEL_KIND	count	비율
C0	503	69.9
C1	1	0.1
C3	7	1.0
T0	16	2.2
T1	18	2.5
T3	2	0.3
T5	43	6.0
T7	35	4.9
T8	95	13.2

가장 많이 **생산**하는 제품 = 'C0'

제품 별 상대 생산율

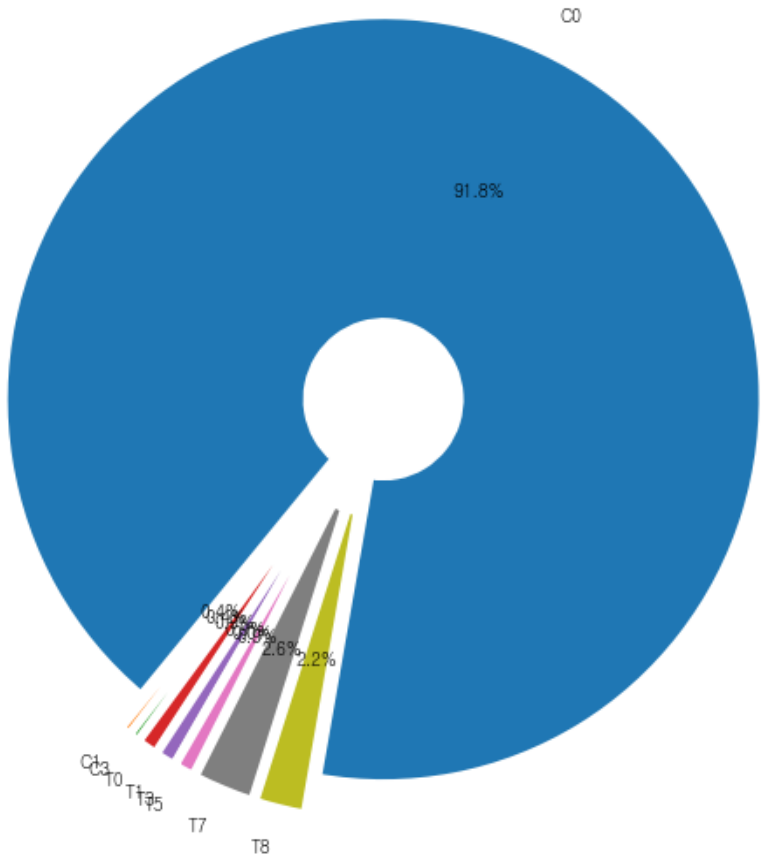


Part 2 # 데이터 분석

- 철강 종류

STEEL_KIND	불량품	불량률
C0	212	91.77
C1	1	0.43
C3	1	0.43
T0	2	0.87
T1	2	0.87
T3	0	0.00
T5	2	0.87
T7	6	2.60
T8	5	2.16

제품 별 상대 불량률



가장 높은 **불량률**을 보이는 제품= 'C0'

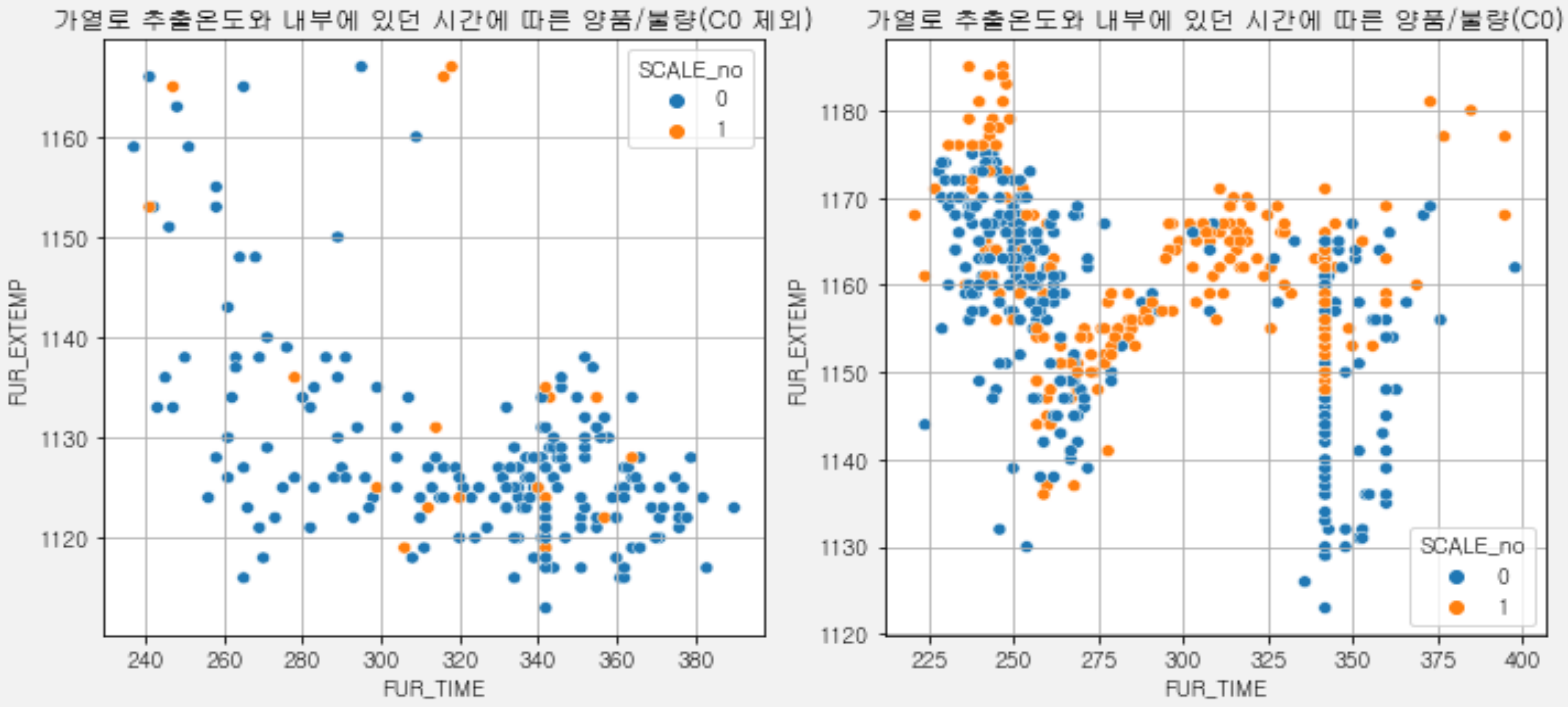
'C0' 를 중점으로 불량품 줄이기

3 'C0' 기준 데이터 시각화

STEP 1		STEP 2		STEP 3
FUR_TIME FUR_EXTEMP	>>	SZ 가열로 온도/시간 HZ 가열로 온도/시간	>>	ROLLING_TEMP ROLLING_DATE

Part 2 데이터 분석

- 'C0' 기준 시각화

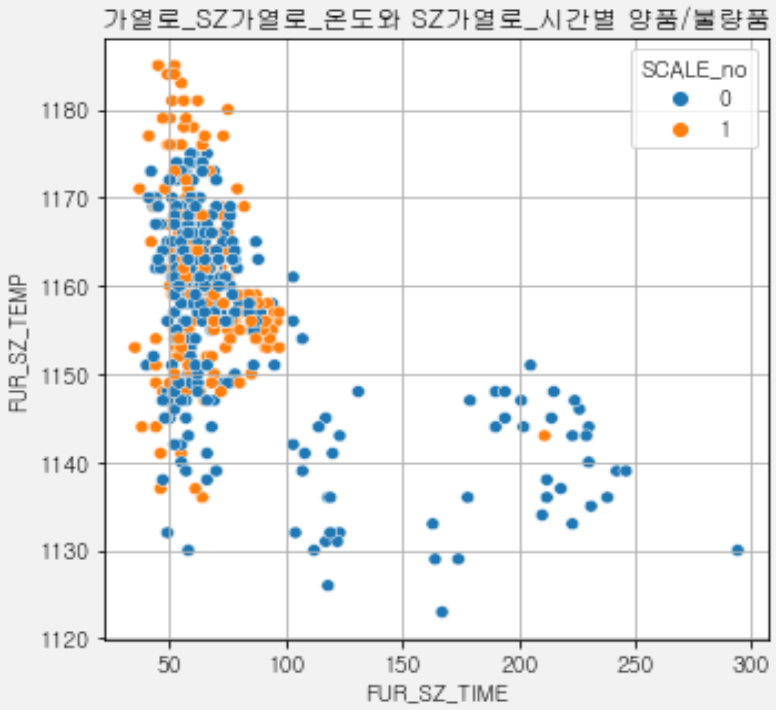


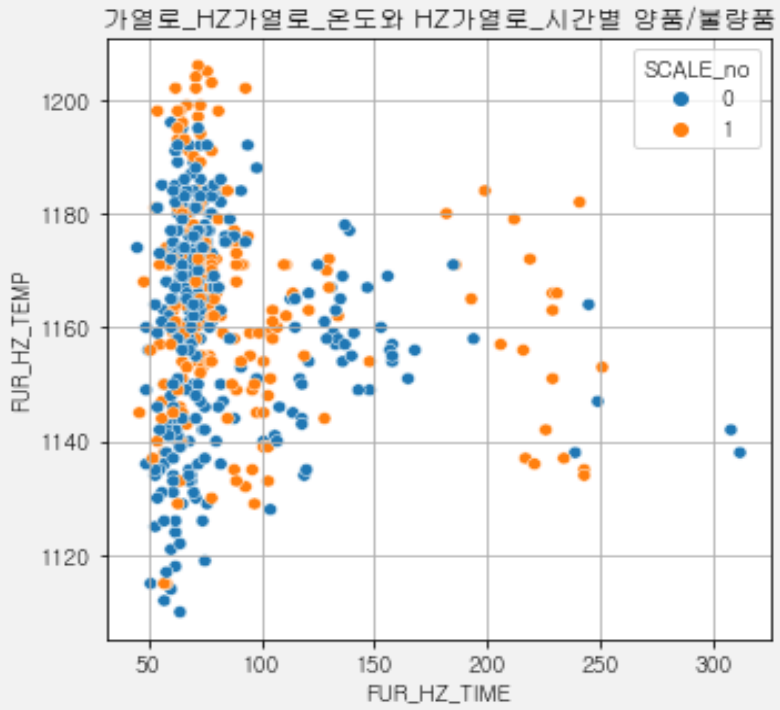
가열로 온도가 1150℃ 이상일 때 불량이 많이 발생하는 것 을 확인

추출 시간이 275초 이상일 때 불량이 많이 발생하는 것을 확인

Part 2 데이터 분석

- 'C0' 기준 시각화



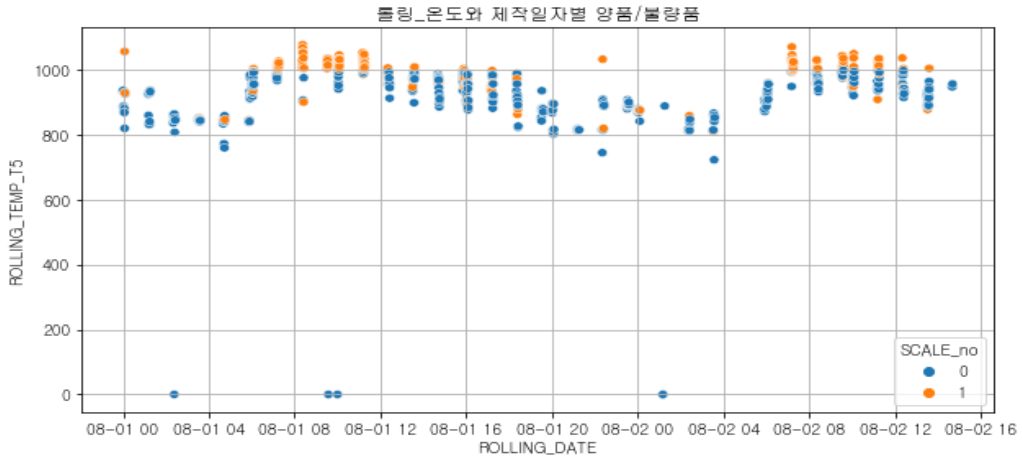


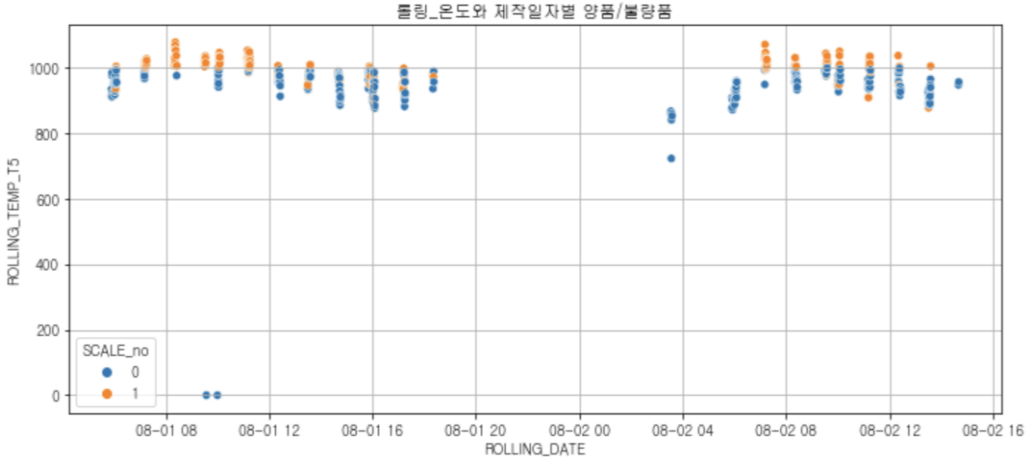
각 가열로별 불량 분포가
동일하지 않다

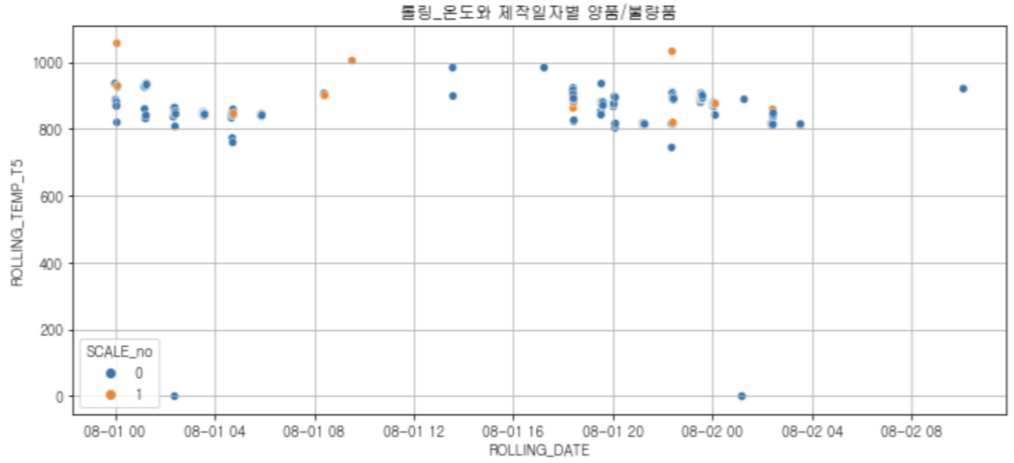
Part 2

데이터 분석

- 'C0' 기준 시각화







롤링 온도가 1000℃ 이상일 때 불량품의 생산량이 증가한다

Part 3

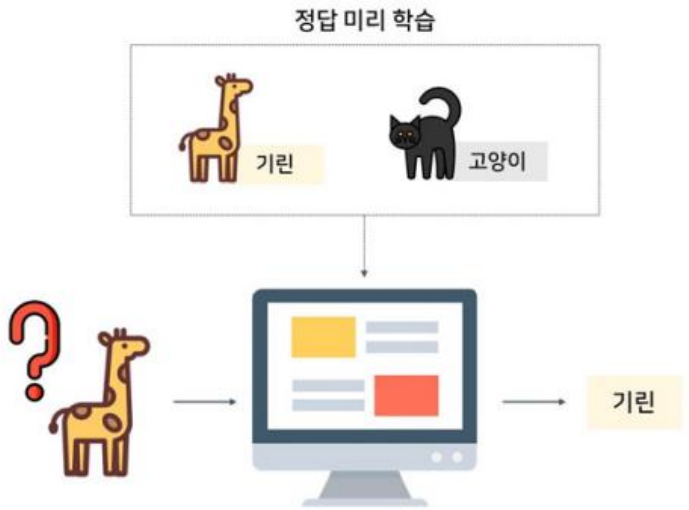
머신러닝 모델

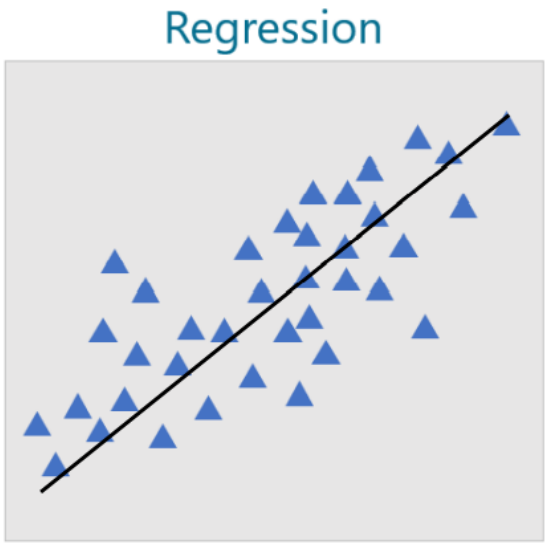
- 예측 모델 선정

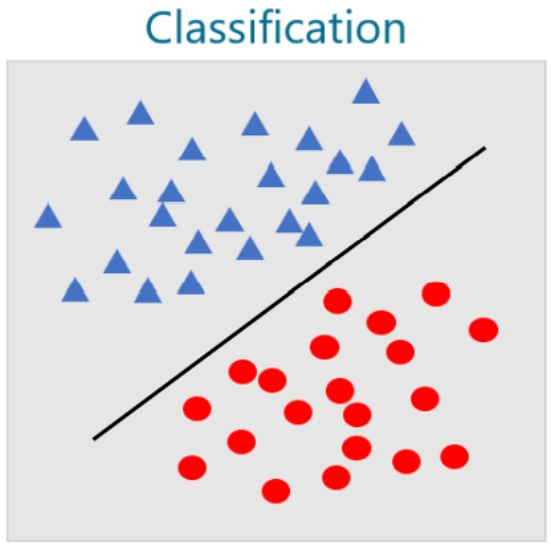
∗ 지도학습 (*Supervised Learning*)

관여자가 문제에 대한 답을 알고 있을 때, 인공지능(*AI*)이 그것을 알아낼 수 있도록 훈련

→







'*C*0' 기준 양품/불량품 분류(*Classification*) 모델 선택

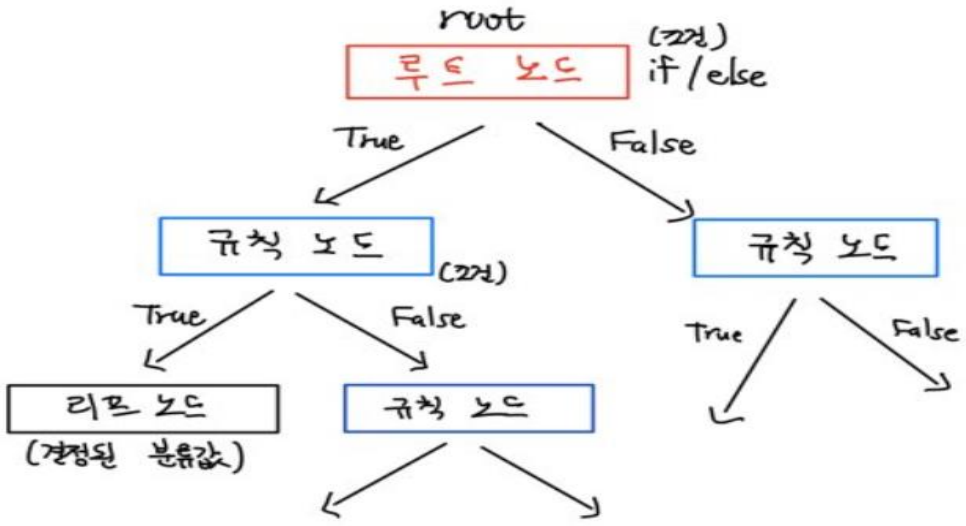
→

Part 3 # 머신러닝 모델

- Decision Tree

Decision Tree

> 뿌리(Root)노드를 기준으로 뻗어나가게 되며,
> 규칙(Decision)노드의 조건에 의해 리프(Leaf)노드로 분류되거나 또다른 규칙노드가 생성



깊이 (depth)

장점

- 데이터 전처리가 다른 모형에 비해 상대적으로 필요 X

- 학습 결과에 대한 이해와 가시적 표현이 쉽다

단점

- 데이터에 크게 의존
(학습 데이터가 바뀌면 구조가 크게 바뀔 수 있음)

- 깊어짐에 따라 분류를 결정하는 방식이 더욱 복잡해짐> 결정트리의 예측 성능 저하

Part 3

머신러닝 모델

- 분류 모델 Decision Tree

```
target= ['HSB', 'SCALE', 'STEEL_KIND', 'rolling_slot',
              'FUR_NO','FUR_HZ_TEMP','FUR_HZ_TIME','FUR_SZ_TEMP','FUR_SZ_TIME','FUR_TIME','ROLLING_TEMP_T5','ROLLING_DESCALING']
dataset = df[target]
dataset = pd.get_dummies(dataset)
drop_dataset = dataset.drop(['SCALE_양품', 'STEEL_KIND_C1','STEEL_KIND_C3','STEEL_KIND_T0','STEEL_KIND_T1','STEEL_KIND_T3',
                             'STEEL_KIND_T5','STEEL_KIND_T7']
                             ,axis = 1)
```

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X = drop_dataset.drop(['SCALE_불량'], axis = 1)
Y = drop_dataset['SCALE_불량'].ravel()

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3, random_state = 0)

model = DecisionTreeClassifier(max_depth = 5)
model.fit(X_train, Y_train)

train_accuracy = model.score(X_train, Y_train)
test_accuracy = model.score(X_test, Y_test)
print(f'훈련 정확도 : {train_accuracy}')
print(f'테스트 정확도 : {test_accuracy}')
```

```
훈련 정확도 : 0.9722222222222222
테스트 정확도 : 0.9675925925925926
```

```
train_accuracy - test_accuracy
```

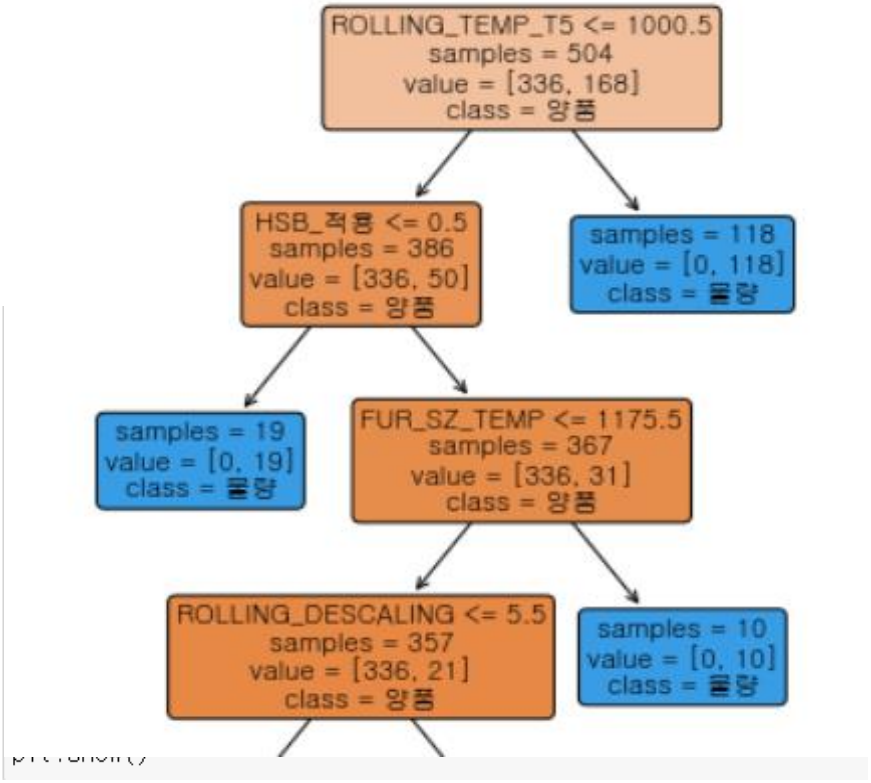
```
0.00462962962962965
```

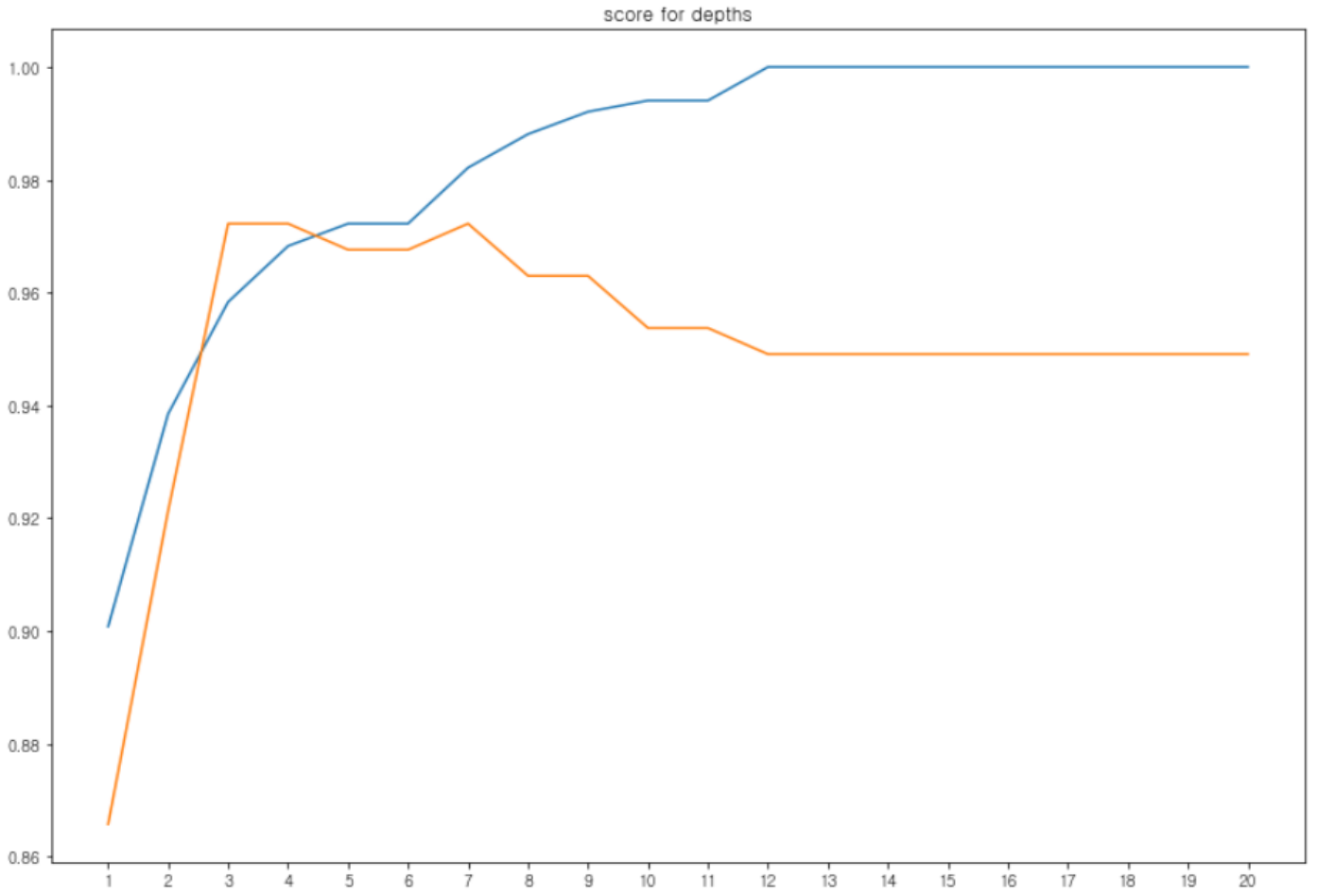
Part 3

머신러닝 모델

- 분류 모델 Decision Tree

Decision Tree 결과 시각화





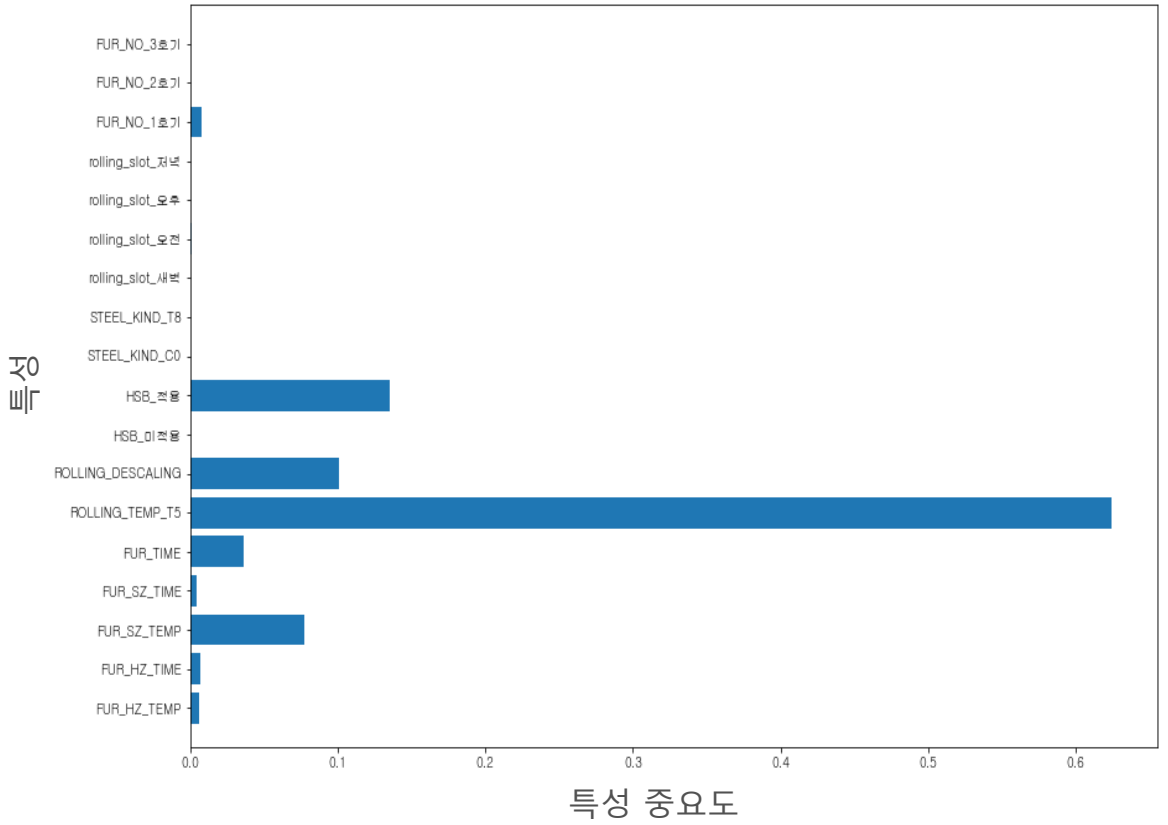
Part 3 머신러닝 모델

- 분류 모델 Decision Tree

```
importance[importance['coefficient']!=0.00].sort_values(by = "coefficient",ascending = False)
```

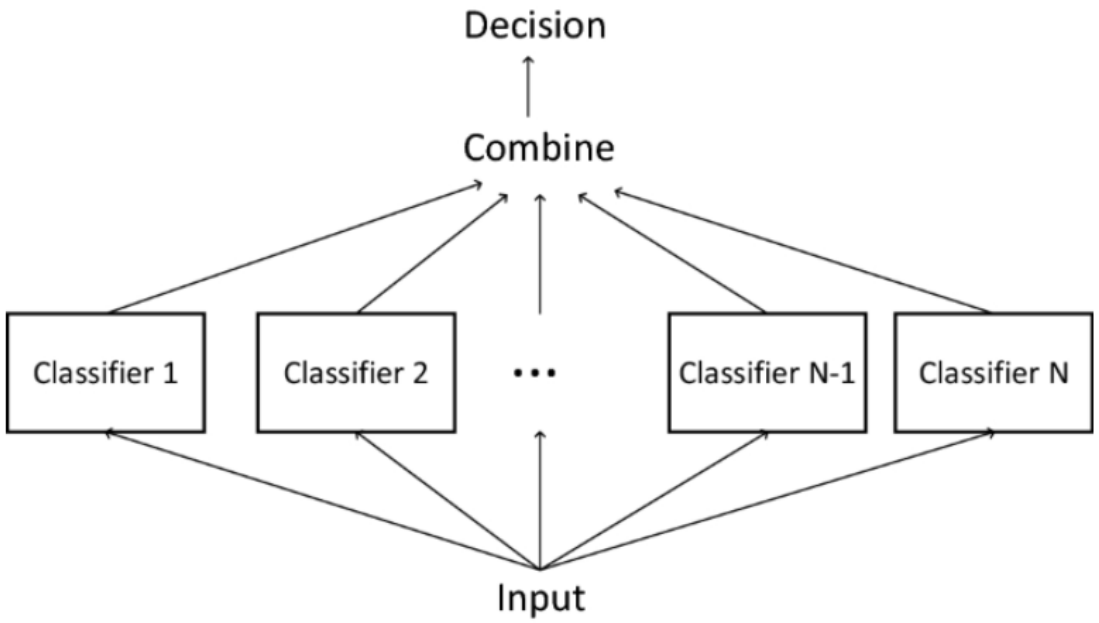
	feature_names	coefficient
5	ROLLING_TEMP_T5	0.694772
8	HSB_적용	0.153631
2	FUR_SZ_TEMP	0.087427
6	ROLLING_DESCALING	0.045576
4	FUR_TIME	0.018595

Decision Tree결과를 통한
특성 중요도 시각화



앙상블 학습 (Ensemble Learning)

> 여러 개의 분류기(Classifier)에서 각각 예측을 수행,



모델 종류

- 보팅 (Voting)
- 배깅 (Bagging)
- 랜덤포레스트 (Random Forest)
- 그레디언트 부스팅 (Gradient Boosting)
- 스테킹 (Stacking)

Part 3 머신러닝 모델

- Random Forest

- **Random Forest** 의사결정 트리의 오버피팅의 한계를 극복하기 위한 전략

- **오버피팅**을 피하기 위해 **임의(random)**의 숲을 구성

- 다수의 나무들로부터 분류를 집계하기 때문에 오버피팅이 나타나는 나무의 영향력을 줄일 수 있음

Part 3 머신러닝 모델

- Random Forest

```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier

X = drop_dataset.drop(['SCALE_불량'], axis = 1)
Y = drop_dataset['SCALE_불량'].ravel()

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3, random_state = 0)
```

```
scaler = StandardScaler()
scaler.fit(X_train)

x_train_std = scaler.transform(X_train)
x_test_std = scaler.transform(X_test)
```

```
rf_clf = RandomForestClassifier(max_depth = 4, n_estimators=150)

rf_clf.fit(x_train_std, Y_train)
```

```
RandomForestClassifier(max_depth=4, n_estimators=150)
```

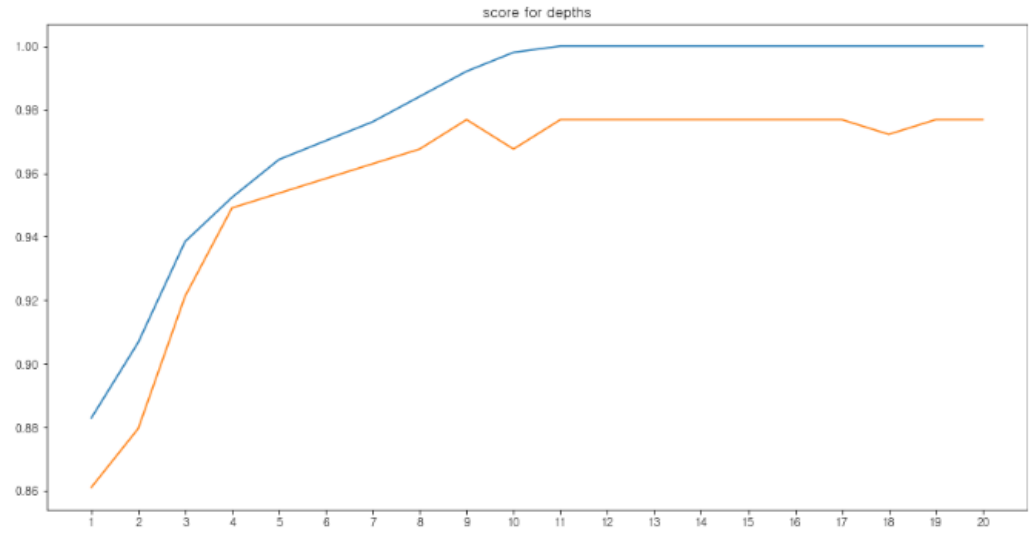
```
rf_train_score = rf_clf.score(x_train_std, Y_train)
rf_test_score = rf_clf.score(x_test_std, Y_test)
print(f'랜덤포레스트 훈련 정확도는 {round(rf_train_score,3)} 입니다.')
print(f'랜덤포레스트 테스트 정확도는 {round(rf_test_score,3)} 입니다.')
```

```
랜덤포레스트 훈련 정확도는 0.952 입니다.
랜덤포레스트 테스트 정확도는 0.949 입니다.
```

```
round(rf_train_score,3) - round(rf_test_score,3)
```

```
0.0030000000000000027
```

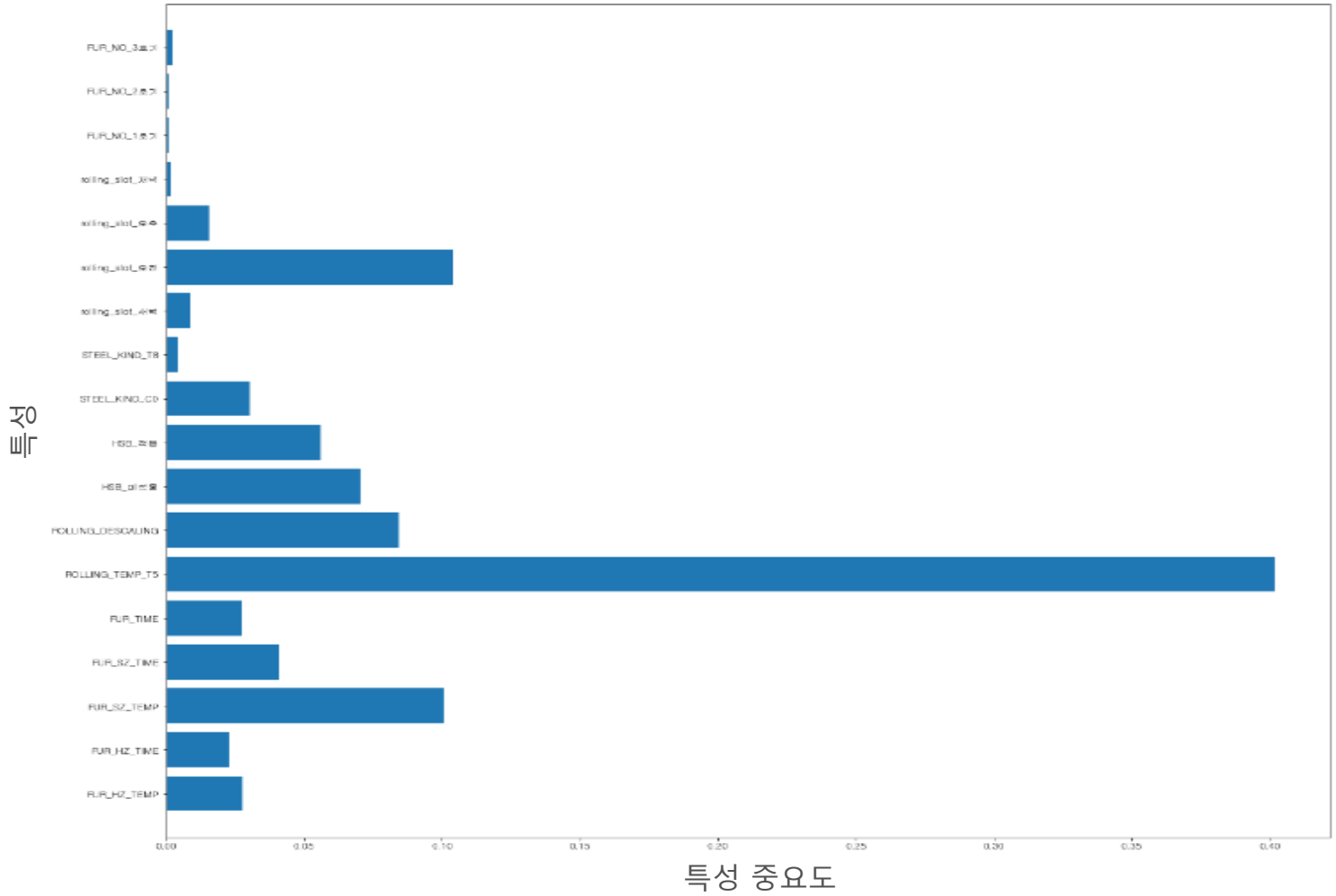
```
train_score=[]
test_score=[]
for i in range(1,21):
    rf_clf = RandomForestClassifier(max_depth = i, n_estimators=150)
    rf_clf.fit(x_train_std, Y_train)
    train_score.append(rf_clf.score(x_train_std, Y_train))
    test_score.append(rf_clf.score(x_test_std, Y_test))
plt.figure()
plt.title('score for depths')
plt.plot(range(1,21),train_score)
plt.plot(range(1,21),test_score)
plt.xticks(range(1,21))
plt.show()
```



Part 3 머신러닝 모델

- Random Forest

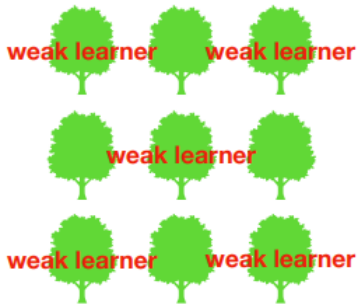
	feature_names	coefficient
5	ROLLING_TEMP_T5	0.401593
12	rolling_slot_오전	0.104162
2	FUR_SZ_TEMP	0.100554
6	ROLLING_DESCALING	0.084385
7	HSB_미적용	0.070386
8	HSB_적용	0.055893
3	FUR_SZ_TIME	0.040705
9	STEEL_KIND_C0	0.030310
0	FUR_HZ_TEMP	0.027710
4	FUR_TIME	0.027326
1	FUR_HZ_TIME	0.022899
13	rolling_slot_오후	0.015511
11	rolling_slot_새벽	0.008819
10	STEEL_KIND_T8	0.004210
17	FUR_NO_3호기	0.002235
14	rolling_slot_저녁	0.001494
16	FUR_NO_2호기	0.000986
15	FUR_NO_1호기	0.000823



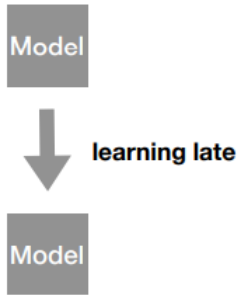
Part 3 # 머신러닝 모델

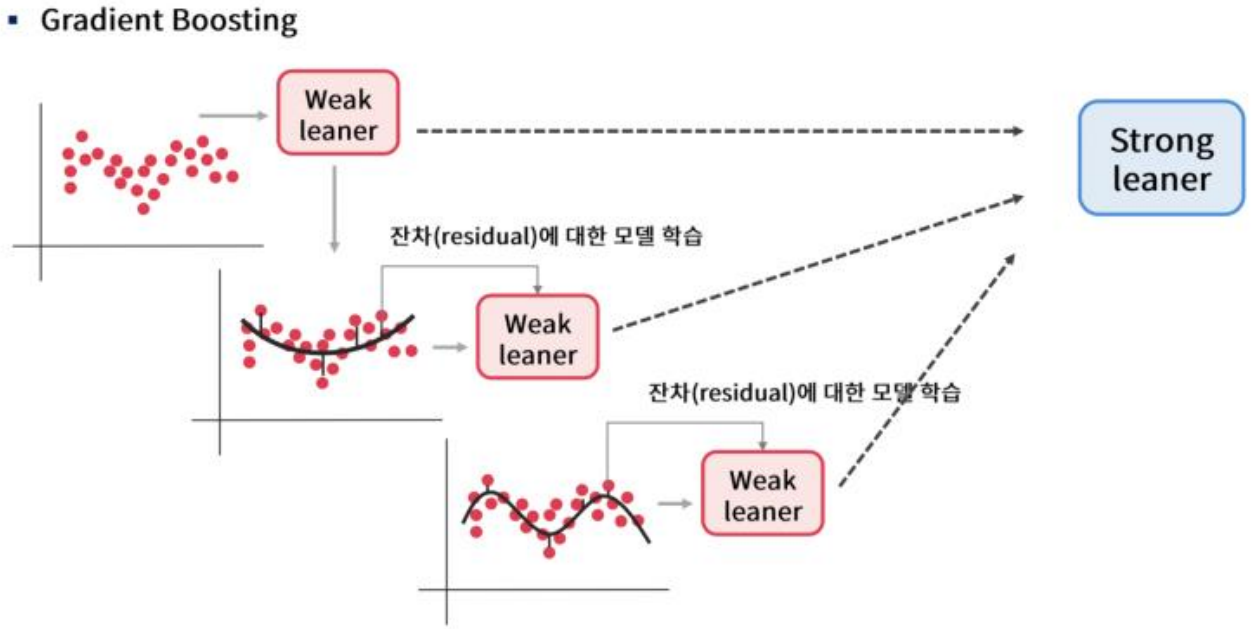
- Gradient Boosting

- **Gradient Boosting**



Gradient Boosting





- 약한 모델들을 단계적으로 **Boosting**하는 과정에서 이전 모델의 오류를 손실함수로 나타내고 이 손실함수를 최소화하는 분석기법

- **Boosting** : 단순한 모델들을 결합하여 단계적으로 학습함으로써 이전 모델의 약점을 점점 보완해 가는 기법

Part 3 머신러닝 모델

- Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier


gbrt = GradientBoostingClassifier(max_depth = 3, learning_rate = 0.1)
gbrt.fit(x_train_std, Y_train)
```

```
GradientBoostingClassifier()
```

```
gbrt_train_score = gbrt.score(x_train_std, Y_train)
gbrt_test_score = gbrt.score(x_test_std, Y_test)
print(f'그레디언트부스팅 훈련 정확도는 {round(gbrt_train_score,3)} 입니다.')
print(f'그레디언트부스팅 테스트 정확도는 {round(gbrt_test_score,3)} 입니다.')
```

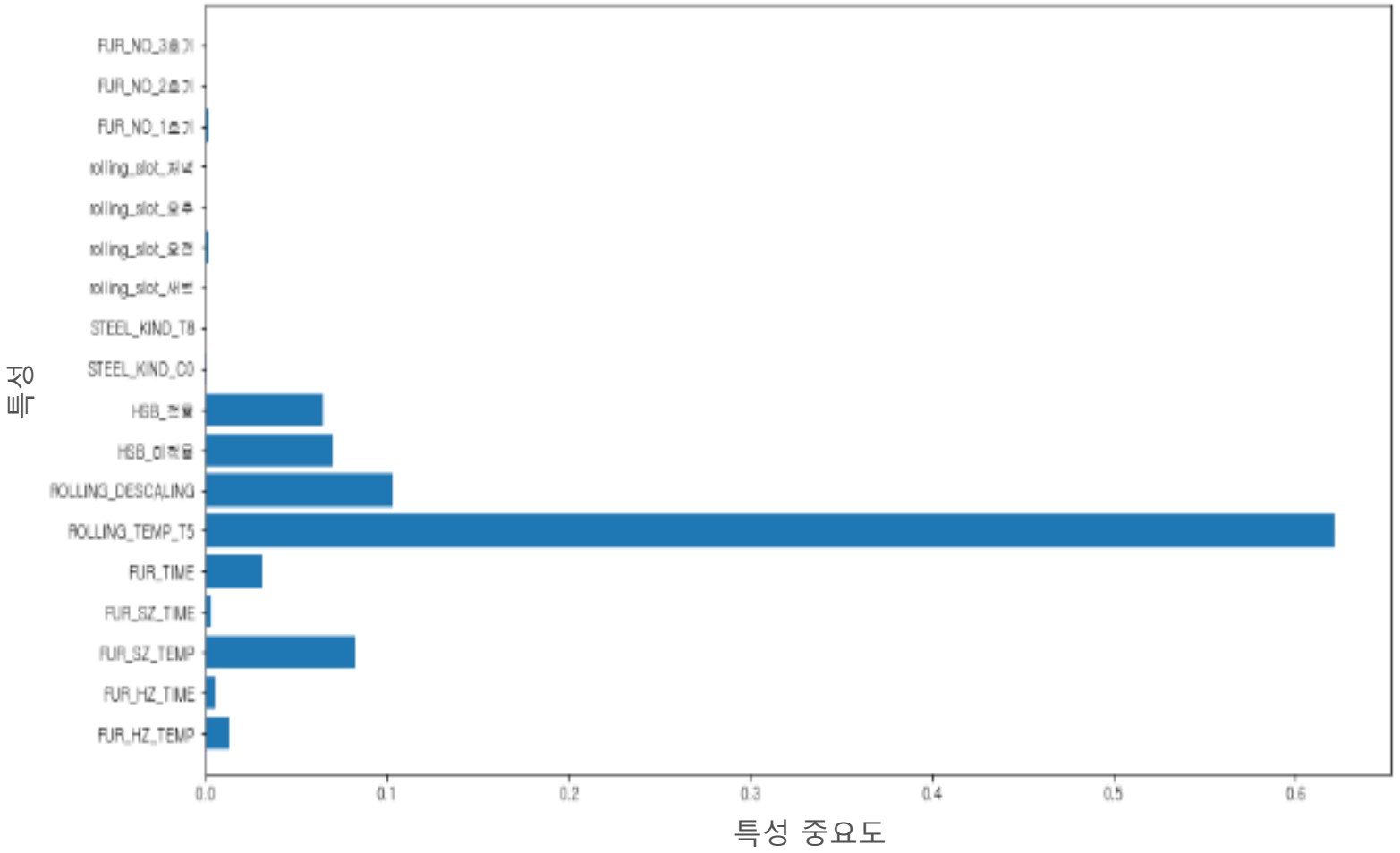
```
그레디언트부스팅 훈련 정확도는 1.0 입니다.
그레디언트부스팅 테스트 정확도는 0.995 입니다.
```

```
train_score=[]
test_score=[]
for i in range(1,21):
    gbrt = GradientBoostingClassifier(max_depth = i, learning_rate = 0.1)
    gbrt.fit(x_train_std, Y_train)
    train_score.append(gbrt.score(x_train_std, Y_train))
    test_score.append(gbrt.score(x_test_std, Y_test))
plt.figure()
plt.title('score for depths')
plt.plot(range(1,21),train_score)
plt.plot(range(1,21),test_score)
plt.xticks(range(1,21))
plt.show()
```

Part 3 머신러닝 모델

- Gradient Boosting

	feature_names	coefficient
5	ROLLING_TEMP_T5	0.621451
6	ROLLING_DESCALING	0.103057
2	FUR_SZ_TEMP	0.082401
7	HSB_미적용	0.070155
8	HSB_적용	0.065040
4	FUR_TIME	0.031466
0	FUR_HZ_TEMP	0.013472
1	FUR_HZ_TIME	0.005971
3	FUR_SZ_TIME	0.003405
15	FUR_NO_1호기	0.001326
12	rolling_slot_오전	0.001315
9	STEEL_KIND_C0	0.000875
17	FUR_NO_3호기	0.000066



Part 3 # 머신러닝 모델

Decision Tree

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, pred_dt)
print(conf_matrix)
```

```
[[145   8]
 [  3  60]]
```

Random Forest

```
conf_matrix = confusion_matrix(Y_test, pred_rf)
print(conf_matrix)
```

```
[[153   0]
 [  5  58]]
```

Gradient Boosting

```
conf_matrix = confusion_matrix(Y_test, pred_gb)
print(conf_matrix)
```

```
[[153   0]
 [  3  60]]
```

각 모델에 대한 Confusion Matrix

```
from sklearn.metrics import classification_report
class_report = classification_report(Y_test, pred_dt)
print(class_report)
```

	precision	recall	f1-score	support
0	0.98	0.95	0.96	153
1	0.88	0.95	0.92	63
accuracy			0.95	216
macro avg	0.93	0.95	0.94	216
weighted avg	0.95	0.95	0.95	216

```
class_report = classification_report(Y_test, pred_rf)
print(class_report)
```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	153
1	1.00	0.92	0.96	63
accuracy			0.98	216
macro avg	0.98	0.96	0.97	216
weighted avg	0.98	0.98	0.98	216

```
class_report = classification_report(Y_test, pred_gb)
print(class_report)
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	153
1	1.00	0.95	0.98	63
accuracy			0.99	216
macro avg	0.99	0.98	0.98	216
weighted avg	0.99	0.99	0.99	216

각 모델에 대한 Classification Report

Part 4

기대 효과

Part 4 참조

https://newsroom.posco.com/kr/%EA%B3%B5%EC%9B%90-%EC%86%8D-%EC%A0%9C%EC%B2%A0%EC%86%8C/

https://www.hankyung.com/economy/article/2020111991421

https://www.hankyung.com/economy/article/2020111991421

https://gomguard.tistory.com/173

https://blog.naver.com/philia38/222881874984

https://iphoong.tistory.com/6

https://communities.sas.com/t5/SAS-Tech-Tip/EM-%EA%B7%B8%EB%9E%98%EB%94%94%EC%96%B8%ED%8A%B8-%EB%B6%80%EC%8A%A4%ED%8C%85-Gradient-Boosting-%EC%9D%B4%EB%A1%A0/ta-p/656745

https://m.blog.naver.com/luvwithcat/222103025023

https://blog.naver.com/teorw272/222082729148

감사합니다

아이구 감사합니다