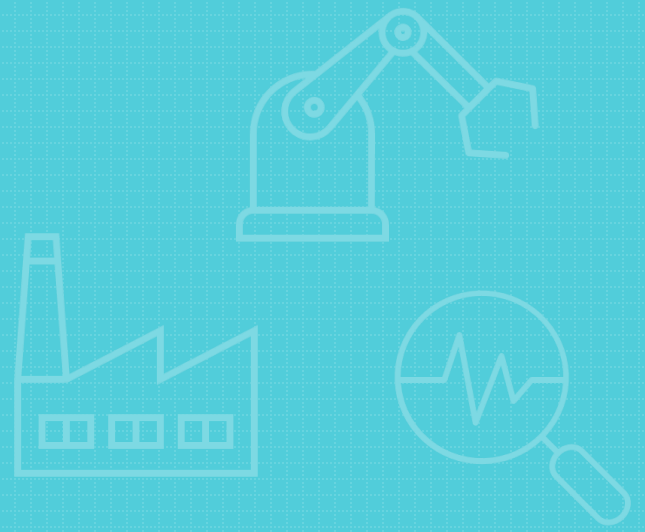
센서 데이터를 활용한
공정 이상 예측



201700638 김비아
201700658 김서희
201701387 박보성
201501257 박세인

01

분석 주제와 분석 목표
분석 효과

02

데이터 소개 및 전처리
▸ 결측치 처리 등

03

EDA
▸ 기술 통계
▸ 라벨 데이터 시각화

04

모델링 및 모델 평가
▸ 로지스틱 회귀

05

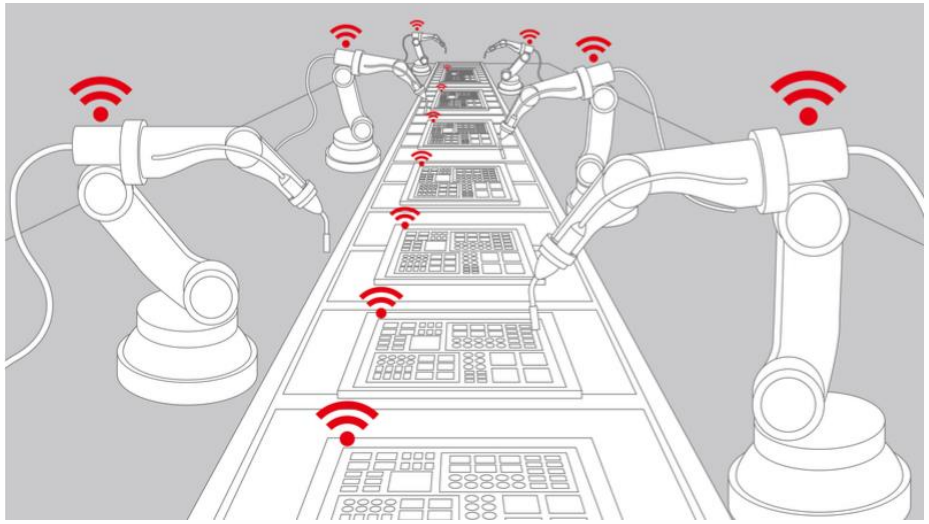
모델 성능 향상

▸ 불균형 데이터 처리: SMOTE 적용
▸ 다양한 데이터마이닝 기법 적용
▸ GridSearch를 통한 하이퍼파라미터 수정
▸ 불균형 데이터 처리: GAN의 적용
▸ GAN + SMOTE

06

모델 선택 및 센서 시각화

분석 주제와 분석 목표



“센서 데이터를 활용한 공정 이상 예측”

공정 이상에 영향을 주는 센서의 발견

이상을 미리 예측할 수 있는 예측력 높은 모델

분석 효과

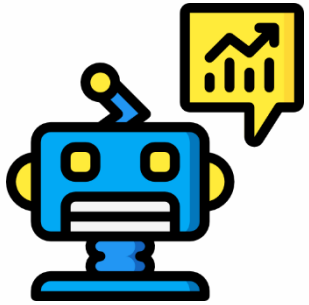
01

사람으로 인한
오류 발생 감소

02

생산 과정에서
인력과 비용 절감

03



공정 이상 예측과
수익 극대화

활용한 모듈

```
# 필요한 모듈 전체 임포트
import warnings
warnings.filterwarnings(action='ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from lightgbm import LGBMRegressor, LGBMClassifier, Booster
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,roc_auc_score
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV
import pandas.testing as tm
import pandas.util.testing as tm
from tgan.model import TGANModel
```

데이터 소개

- 모든 튜플이 0이거나 공백으로 채워진 열을 엑셀 파일에서 수작업으로 제거
- 이를 uci-secom2.csv로 저장하고 데이터 불러옴 `data = pd.read_csv('uci-secom2.csv')`
- 총 592개의 열에서 447개로 감소한 것을 확인 `data.head(6)`

시간 | 0 ~ 589번 센서 | 라벨

	Time	0	1	2	3	4	6	7	8	9	...	581	582	583	584	585	586	587	588	589	Pass/Fail
0	2008-07-19 11:55	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	...	NaN	0.5005	0.0118	0.0035	2.3630	NaN	NaN	NaN	NaN	-1
1	2008-07-19 12:32	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	...	208.2045	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201	0.0060	208.2045	-1
2	2008-07-19 13:17	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	...	82.8602	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484	0.0148	82.8602	1
3	2008-07-19 14:43	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0.0124	...	73.8432	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149	0.0044	73.8432	-1
4	2008-07-19 15:22	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0.0031	...	NaN	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149	0.0044	73.8432	-1
5	2008-07-19 17:53	2946.25	2432.84	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5287	0.0167	...	44.0077	0.4949	0.0189	0.0044	3.8276	0.0342	0.0151	0.0052	44.0077	-1

6 rows × 447 columns

데이터 전처리

➢ -1과 1로 존재하는 라벨을 0과 1로 변경

```
data.loc[(data['Pass/Fail'] == -1), 'Pass/Fail']=0
```

시간 | 0 ~ 589번 센서 | 라벨 (1)

	Time	0	1	2	3	4	6	7	8	9	...	581	582	583	584	585	586	587	588	589	Pass/Fail
0	2008-07-19 11:55	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	...	NaN	0.5005	0.0118	0.0035	2.3630	NaN	NaN	NaN	NaN	-1
1	2008-07-19 12:32	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	...	208.2045	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201	0.0060	208.2045	-1
2	2008-07-19 13:17	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	...	82.8602	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484	0.0148	82.8602	1
3	2008-07-19 14:43	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0.0124	...	73.8432	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149	0.0044	73.8432	-1
4	2008-07-19 15:22	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0.0031	...	NaN	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149	0.0044	73.8432	-1
5	2008-07-19 17:53	2946.25	2432.84	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5287	0.0167	...	44.0077	0.4949	0.0189	0.0044	3.8276	0.0342	0.0151	0.0052	44.0077	-1

6 rows × 447 columns

데이터 전처리

- -1과 1로 존재하는 라벨을 0과 1로 변경 `data.loc[(data['Pass/Fail'] == -1), 'Pass/Fail']=0`
- 결측치를 0으로 대체 `data = data.replace(np.NaN, 0)`

2

시간 | 0 ~ 589번 센서 | 라벨

	Time	0	1	2	3	4	6	7	8	9	...	581	582	583	584	585	586	587	588	589	Pass/Fail
0	2008-07-19 11:55	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	...	NaN	0.5005	0.0118	0.0035	2.3630	NaN	NaN	NaN	NaN	-1
1	2008-07-19 12:32	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	...	208.2045	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201	0.0060	208.2045	-1
2	2008-07-19 13:17	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	...	82.8602	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484	0.0148	82.8602	1
3	2008-07-19 14:43	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0.0124	...	73.8432	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149	0.0044	73.8432	-1
4	2008-07-19 15:22	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0.0031	...	NaN	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149	0.0044	73.8432	-1
5	2008-07-19 17:53	2946.25	2432.84	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5287	0.0167	...	44.0077	0.4949	0.0189	0.0044	3.8276	0.0342	0.0151	0.0052	44.0077	-1

6 rows × 447 columns

데이터 전처리

- -1과 1로 존재하는 라벨을 0과 1로 변경 `data.loc[(data['Pass/Fail'] == -1), 'Pass/Fail']=0`
- 결측치를 0으로 대체 `data = data.replace(np.NaN, 0)`
- 시간 정보 삭제 `data = data.drop(columns = ['Time'], axis = 1)`

3 시간 | 0 ~ 589번 센서 | 라벨

	Time	0	1	2	3	4	6	7	8	9	...	581	582	583	584	585	586	587	588	589	Pass/Fail
0	2008-07-19 11:55	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	...	NaN	0.5005	0.0118	0.0035	2.3630	NaN	NaN	NaN	NaN	-1
1	2008-07-19 12:32	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	...	208.2045	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201	0.0060	208.2045	-1
2	2008-07-19 13:17	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	...	82.8602	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484	0.0148	82.8602	1
3	2008-07-19 14:43	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0.0124	...	73.8432	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149	0.0044	73.8432	-1
4	2008-07-19 15:22	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0.0031	...	NaN	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149	0.0044	73.8432	-1
5	2008-07-19 17:53	2946.25	2432.84	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5287	0.0167	...	44.0077	0.4949	0.0189	0.0044	3.8276	0.0342	0.0151	0.0052	44.0077	-1

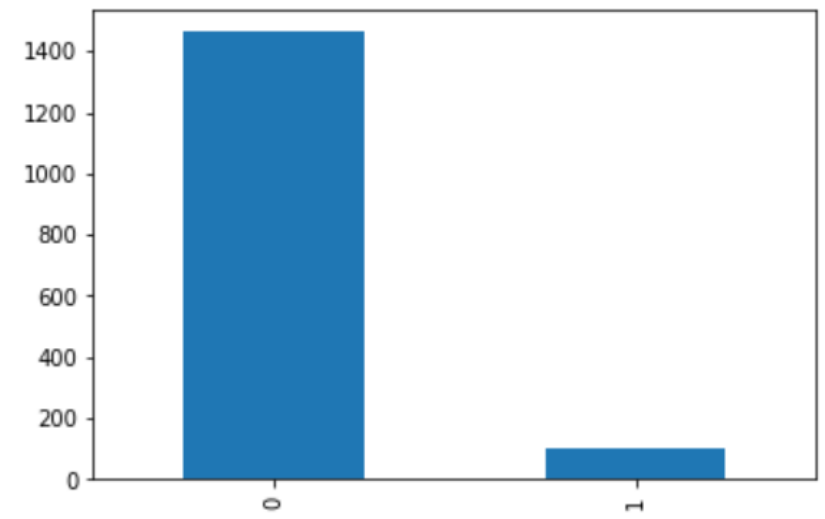
6 rows × 447 columns

EDA

➢ 라벨 분포 확인

```
data['Pass/Fail'].value_counts().plot(kind='bar')
data['Pass/Fail'].value_counts()
```

```
0    1463
1     104
Name: Pass/Fail, dtype: int64
```



- 라벨의 분포가 imbalanced data
- 모델 성능에 영향을 미칠 것으로 예상 가능

- **오버샘플링 방법: SMOTE**

데이터의 개수가 적은 클래스의 표본을 가져온 뒤
임의의 값을 추가하여 새로운 샘플을 만들어 추가

- **오버샘플링 방법: GAN (적대적 생성 신경망)**

생성자 모델과 판별자 모델의 대립과 상호작용을 통해
실제 데이터와 유사한 데이터를 생성하는 딥러닝 알고리즘

모델링 및 모델 평가

- 모델링을 위한 함수 구현
  - modeling (model, x_train, x_test, y_train, y_test)
  - metrics (y_test, pred)

```
def modeling(model, x_train, x_test, y_train, y_test):
    model.fit(x_train, y_train)     # 데이터를 학습시킬 때는 fit 함수를 사용
    pred = model.predict(x_test)
    metrics(y_test,pred)

def metrics(y_test, pred):
    accuracy = accuracy_score(y_test,pred)
    precision = precision_score(y_test,pred) #zero_division=1
    recall = recall_score(y_test,pred)
    f1 = f1_score(y_test,pred)
    roc_score = roc_auc_score(y_test,pred,average='macro')
    print('정확도 : {0:.2f}, 정밀도 : {1:.2f}, 재현율 : {2:.2f}'.format(accuracy,precision,recall))
    print('f1-score : {0:.2f}, auc : {1:.2f}'.format(f1,roc_score,recall))
```

모델링 및 모델 평가

1 x (feature) : y (label)

```
x = data.drop(columns = ['Pass/Fail'], axis = 1)
y = data['Pass/Fail']
y = y.to_numpy().ravel()
```

2 train (8) : test (2)

```
x_train, x_test, y_train, y_test =
train_test_split(x, y, test_size = 0.2, random_state = 486)
```

3 정규화 (StandardScaler)

```
# from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

# x_train에 있는 데이터에 맞춰 정규화를 진행합니다.

x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
x_train, x_test
```

➢ 로지스틱 회귀

```
LR = LogisticRegression(max_iter = 5000)
modeling(LR, x_train, x_test, y_train, y_test)
pred = LR.predict(x_test)
```

(%)	정확도	정밀도	재현율	F1 score	Auc
LR	89	11	10	11	52
		□	□	□	

➢ 낮은 성능 (정밀도, 재현율, F1 score)

➢ 모델 성능 향상을 위한 다양한 기법 적용

모델 성능 향상

1 불균형 데이터 처리 - SMOTE

Q. 라벨의 불균형 문제로 인해 성능이 낮은가?

2 다양한 데이터 마이닝 기법 적용

Q. 로지스틱 회귀보다 성능 좋은 모델링 기법의 탐색

3 GridSearch를 통한 하이퍼파라미터 수정

Q. 선정한 모델의 성능을 더 향상시킬 수 있을까?

4 불균형 데이터 처리 - GAN

Q. GAN을 통한 라벨의 불균형 문제 해결

5 GAN + SMOTE

라벨의 불균형 문제 해결을 위한 모델 중첩 적용으로 성능 향상

모델 성능 향상

1 불균형 데이터 처리 - SMOTE

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=486)

x_train_over, y_train_over = smote.fit_resample(x_train, y_train)

print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', x_train.shape, y_train.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ', x_train_over.shape, y_train_over.shape)
print()
print('SMOTE 적용 후 레이블 값 분포: ₩n', pd.Series(y_train_over).value_counts())
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트:  (1253, 445) (1253,)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트:  (2338, 445) (2338,)

SMOTE 적용 후 레이블 값 분포:
 1    1169
0     1169
dtype: int64
```

➢ 로지스틱 회귀 & SMOTE `modeling(LR, x_train_over, x_test, y_train_over, y_test)`

(%)	정확도도	정밀도도	재현율	F1 score	Auc
LR	89	11	10	11	52
LR SMOTE	84	11	20	14	54

➢ 여전히 낮은 정밀도, 재현율, F1 score

➢ 다양한 데이터 마이닝 기법 적용

모델 성능 향상

2 다양한 데이터 마이닝 기법 적용

- LR
- KNN
- 의사결정나무
- 나이브베이즈
- 랜덤포레스트
- SVC
- XGB
- LGBM

```
models = []
models.append(('LR', LogisticRegression(max_iter =5000)))  #로지스틱 회귀모델
models.append(('KNN', KNeighborsClassifier()))             # KNN 모델
models.append(('CART', DecisionTreeClassifier()))          # 의사결정트리 모델
models.append(('NB', GaussianNB()))                        # 가우시안 나이브 베이즈 모델
models.append(('RF', RandomForestClassifier()))            # 랜덤포레스트 모델
models.append(('SVM', SVC(gamma='auto')))                  # SVM 모델
models.append(('XGB', XGBClassifier()))                    # XGB 모델
models.append(('LGBM', LGBMClassifier()))                  # LGBM 모델
```

➢ 모든 모델

```
for name, model in models:
    model.fit(x_train, y_train)
    msg = "%s - train_score : %f, test score : %f" % (name, model.score(x_train, y_train), model.score(x_test, y_test))
    print(msg)
```

➢ 모든 모델 & SMOTE

```
for name, model in models:
    model.fit(x_train_over, y_train_over)
    msg = "%s - train_score : %f, test score : %f" % (name, model.score(x_train_over, y_train_over), model.score(x_test, y_test))
    print(msg)
```

모델 성능 향상

2 다양한 데이터 마이닝 기법 적용

➢ 모든 모델

(%)	SMOTE 적용 전	
Model	Train	Test
LR	97	89
KNN	93	93
CART	100	87
NB	28	27
RF	99	93
SVM	93	93
XGB	100	93
LGBM	100	93

➢ 모든 모델 & SMOTE

SMOTE 적용 후	
Train	Test
98	84
68	33
100	86
64	30
100	93
99	92
100	93
100	93

➢ 크게 유의미하지 않음
(KNN처럼 떨어진 경우도 有)

➢ **SMOTE 적용 전 vs. 후**

- 적용 전 최고치: 93.63
- 적용 후 최고치: 93.31

➢ SMOTE 적용 전 RF 선택
(RF, SVM, XGB, LGBM 동일)

모델 성능 향상

3 GridSearch를 통한 하이퍼파라미터 수정

```
params = { 'n_estimators' : [10, 200],
           'max_depth' : [10, 15, 20],
           'min_samples_leaf' : [6, 8, 10],
           'min_samples_split' : [6, 8, 10]
            }

# RandomForestClassifier 객체 생성 후 GridSearchCV 수행

rf_clf = RandomForestClassifier(random_state = 486, n_jobs = -1)
grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 5, n_jobs = -1)
grid_cv.fit(x_train, y_train)
```

- 최적 하이퍼 파라미터: max_depth = 10, min_samples_leaf = 6, min_samples_split = 6, n_estimators = 200
- 최고 예측 정확도: 0.9330

```
rf_clf1 = RandomForestClassifier(n_estimators = 200,
                                 max_depth = 10,
                                 min_samples_leaf = 6,
                                 min_samples_split = 6,
                                 random_state = 486,
                                 n_jobs = -1)

rf_clf1.fit(x_train, y_train)
pred = rf_clf1.predict(x_test)
print('예측 정확도: {:.4f}'.format(accuracy_score(y_test,pred)))
```

- 하이퍼파라미터 조정 전 랜덤 포레스트와 동일한 성능

Model	Test
RF	93.63

=

Test
93.63

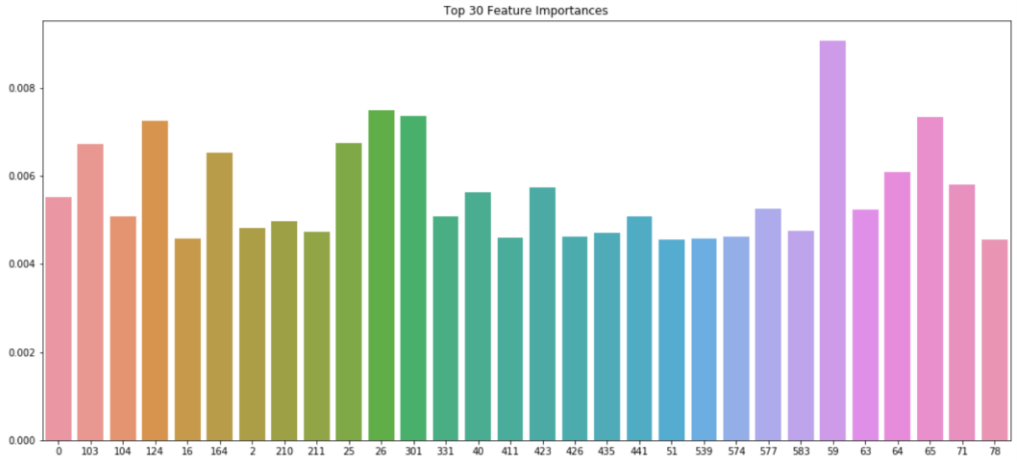
모델 성능 향상

4 불균형 데이터 처리 - GAN

- 중요 변수 30개만 포함한 데이터 (uci-secom3)으로 GAN 실행하여 tgan_data 생성
- Tgan_data 중 라벨 1인 데이터 (Fail_data)와 uci-secom3을 합쳐 uci-secom4를 생성

* 최종 선택한 모델(RF) 기준

```
RF = RandomForestClassifier().fit(x_train,y_train)

%matplotlib inline
feature_names = x.columns.tolist()

ftr_importances_values = RF.feature_importances_
ftr_importances = pd.Series(ftr_importances_values, index = feature_names)
ftr_top30 = ftr_importances.sort_values(ascending=False)[:30]

plt.figure(figsize=(18,8))
plt.title('Top 30 Feature Importances')
sns.barplot(x=ftr_top30.index, y=ftr_top30)
plt.show()
```



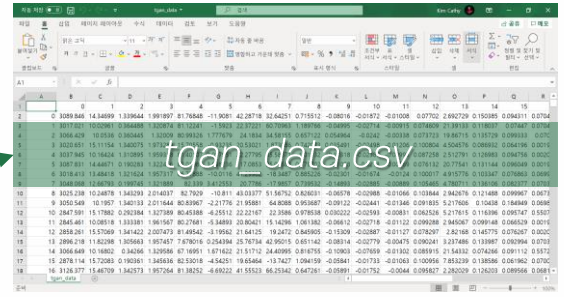
모델 성능 향상

4 불균형 데이터 처리 - GAN

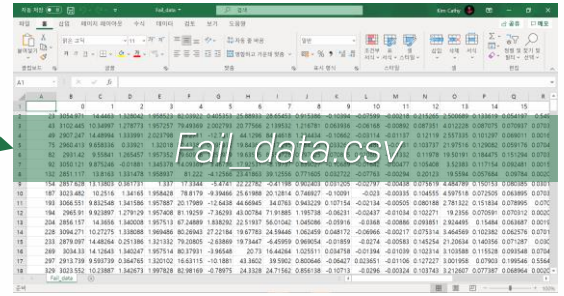
```
# 중요도 top 30의 변수만 포함한 데이터 파일
# 모듈을 작동시키는데 시간이 오래 걸려 데이터양을 줄임
df = pd.read_csv('uci-secom3.csv')
df = df.replace(np.NaN, 0)
df.loc[(df['Pass/Fail'] == -1), 'Pass/Fail']=0
df = df.drop(columns = ['Time'], axis = 1)

# uci-secom3 파일에 대한 tgan 실행
# Tabular data에 대한 gan을 실행시켜주는 모듈
# 참고주소 : https://github.com/sdv-dev/TGAN
df_columns = df.columns
continuous_columns = [df.columns.get_loc(c) for c in data.select_dtypes(include=['float']).columns]

tf.reset_default_graph() # 모듈이 여러번 실행됐을 시 변수가 중복되어 발생하는 오류 해결
tgan = TGANModel(continuous_columns, batch_size = 50 )
tgan.fit(df)

# 만들어진 모델 경로에 저장
model_path = 'models/mymodel2.pkl'
tgan.save(model_path)

#만들어진 모델 csv 파일로 저장
new_tgan = TGANModel.load(model_path)
new_tgan.to_csv('tgan_data.csv')

#만들어진 데이터 중 Pass/Fail 열(tgan 데이터에서는 30번 열)이 1인 데이터만 저장
new_data = new_tgan[new_tgan[30] ==1]
new_data.to_csv('Fail_data.csv')
```

열 번호: RF 중요변수 오름차순

행 번호: tgan이 자동으로 부여



tgan_data 중 라벨 1만 추출



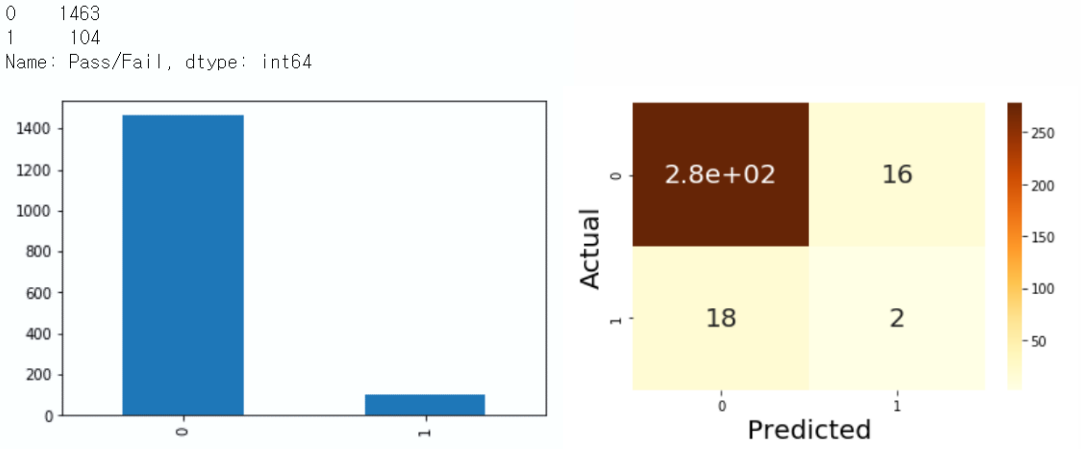
- uci-secom4: 전처리, x:y, train:test, 정규화 (생략)

모델 성능 향상

4 불균형 데이터 처리 - **GAN**

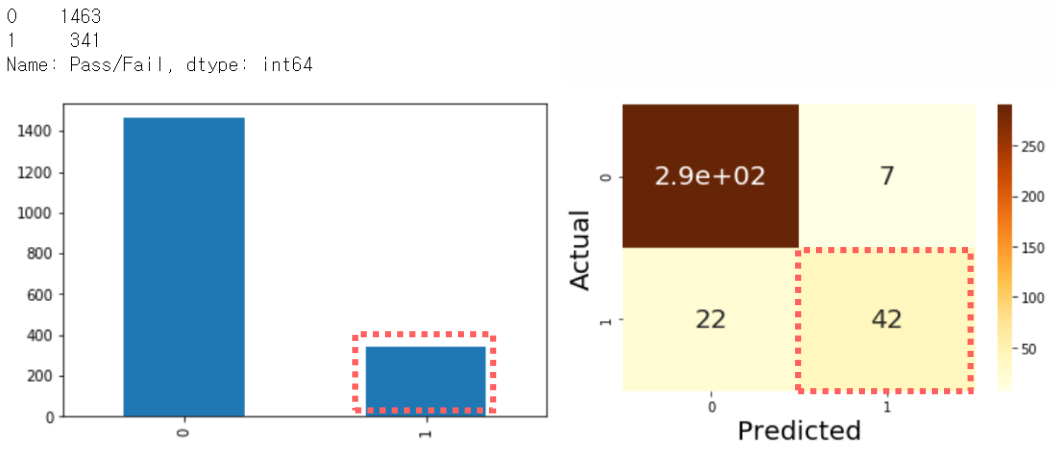
➢ 기존 로지스틱 회귀

(%)	정확도	정밀도	재현율	F1 score	Auc
LR	89	11	10	11	52



➢ GAN 로지스틱 회귀

(%)	정확도	정밀도	재현율	F1 score	Auc
LR GAN	92	86	66	74	82



모델 성능 향상

5 GAN + SMOTE

```
smote = SMOTE(random_state=486)

Gan_x_train_over, Gan_y_train_over = smote.fit_resample(Gan_x_train, Gan_y_train)

print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', Gan_x_train.shape, Gan_y_train.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ', Gan_x_train_over.shape, Gan_y_train_over.shape)
print()
print('SMOTE 적용 후 레이블 값 분포: ₩n', pd.Series(Gan_y_train_over).value_counts())
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트:  (1443, 30) (1443,)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트:  (2332, 30) (2332,)

SMOTE 적용 후 레이블 값 분포:
 1    1166
0    1166
Name: Pass/Fail, dtype: int64
```

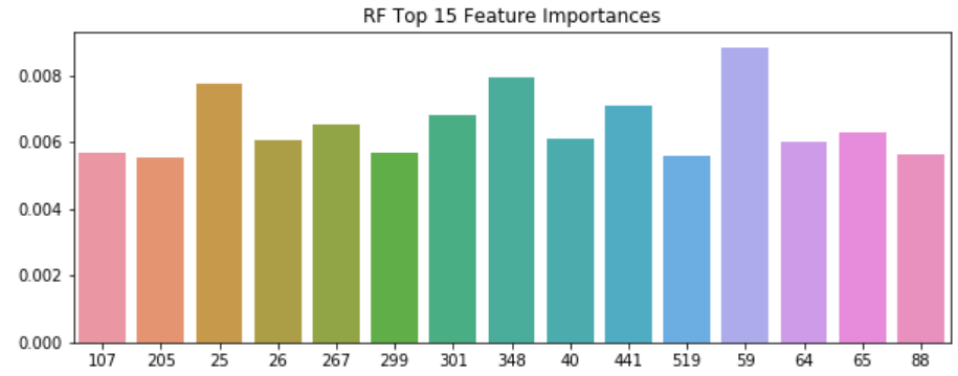
모델 성능 향상

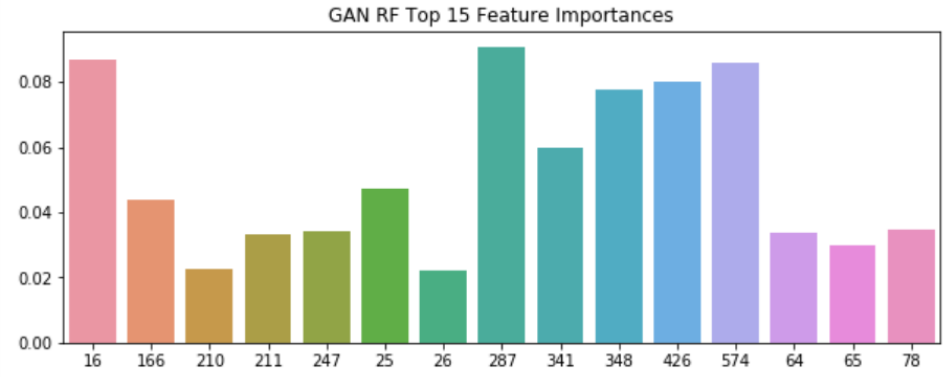
5 GAN + SMOTE

	기존 데이터				GAN 데이터			
(%)	SMOTE 적용 전		SMOTE 적용 후		SMOTE 적용 전		SMOTE 적용 후	
Model	Train	Test	Train	Test	Train	Test	Train	Test
LR	97	89	98	84	93	91	85	84
KNN	93	93	68	33	92	91	94	80
CART	100	87	100	86	100	87	100	80
NB	28	27	64	30	89	87	81	83
RF	99	93	100	93	100	93	100	91
SVM	93	93	99	92	94	92	91	86
XGB	100	93	100	93	100	91	100	91
LGBM	100	93	100	93	100	93	100	92

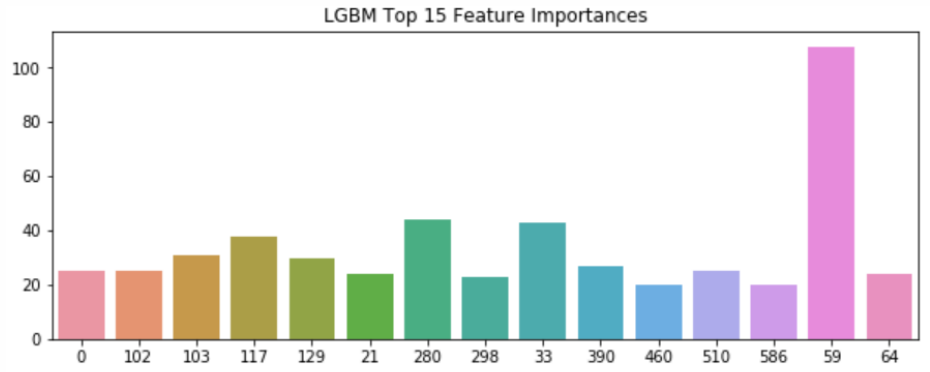
모델 선택 및 센서 시각화

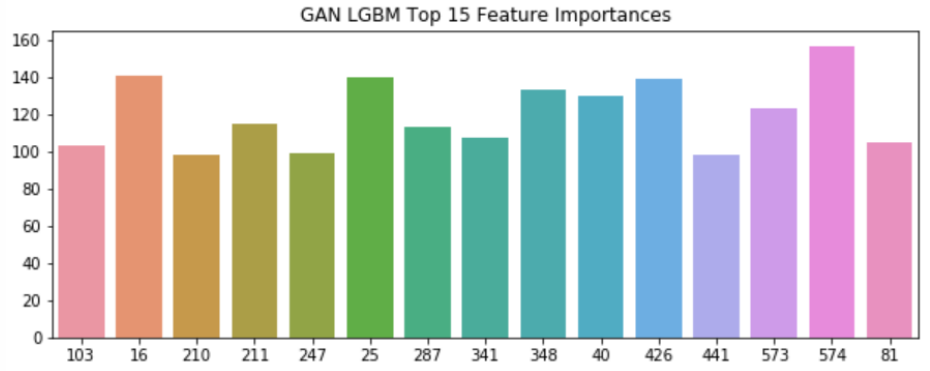
➢ 랜덤 포레스트 (93.63%)





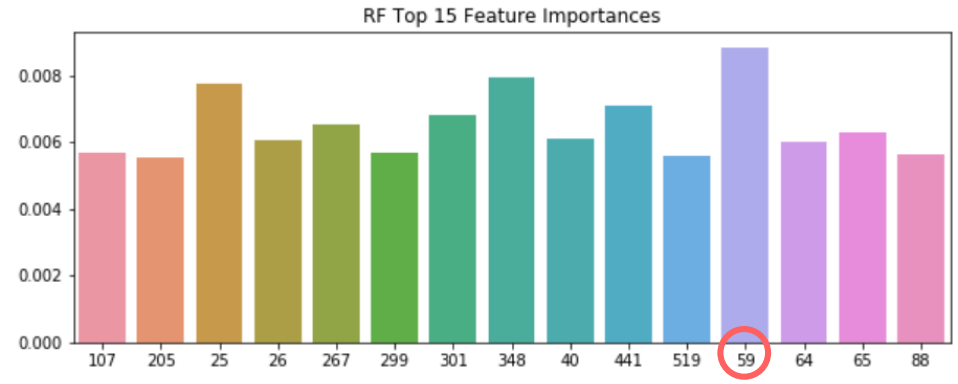
➢ LGBM (93.63%)



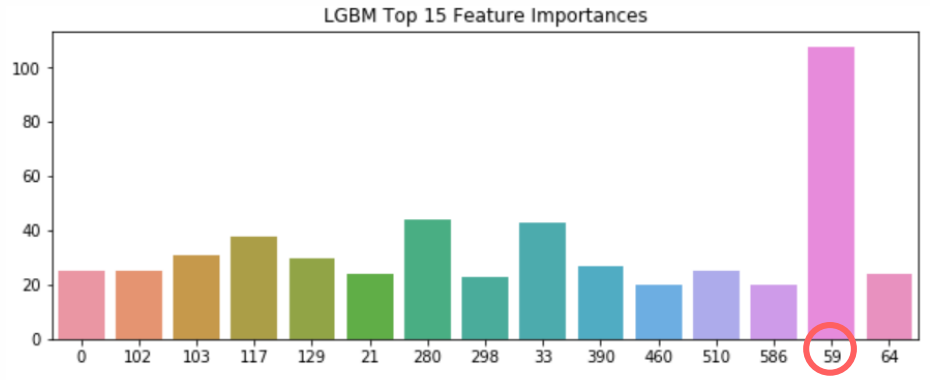


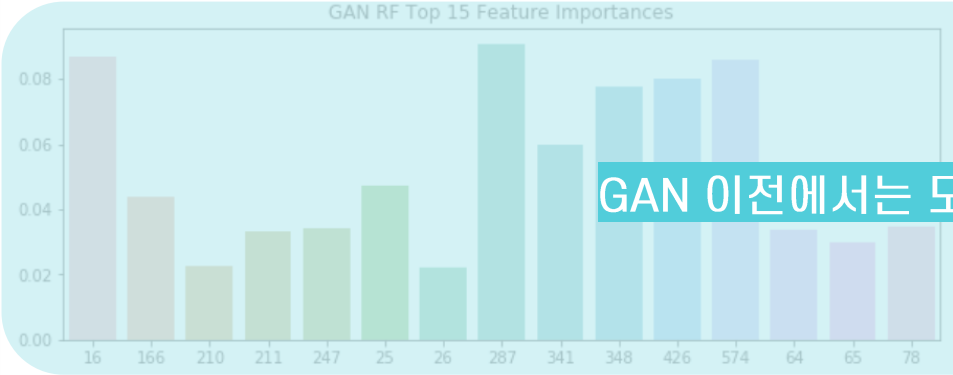
모델 선택 및 센서 시각화

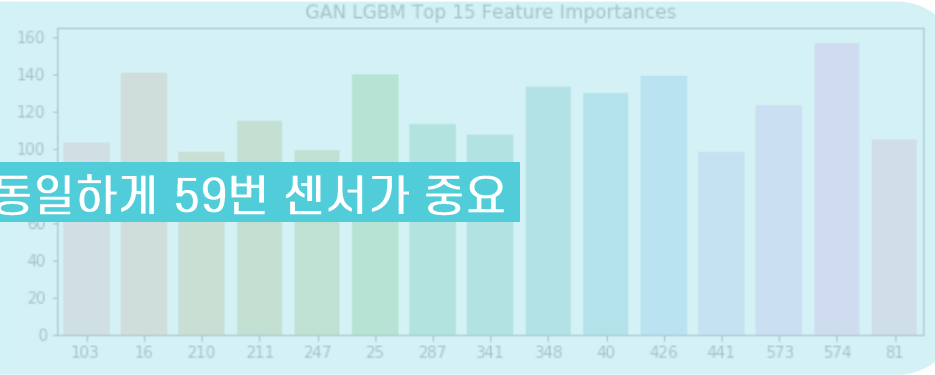
➢ 랜덤 포레스트 (93.63%)



➢ LGBM (93.63%)



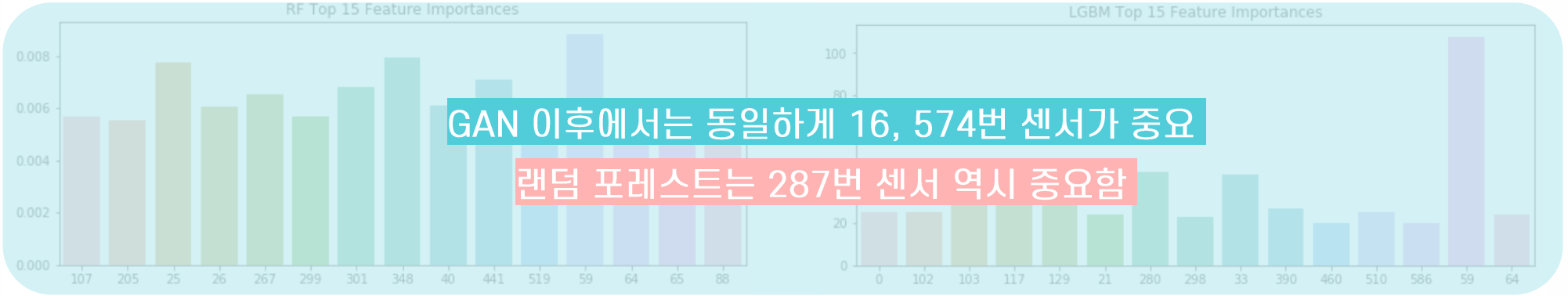




GAN 이전에서는 모두 동일하게 59번 센서가 중요

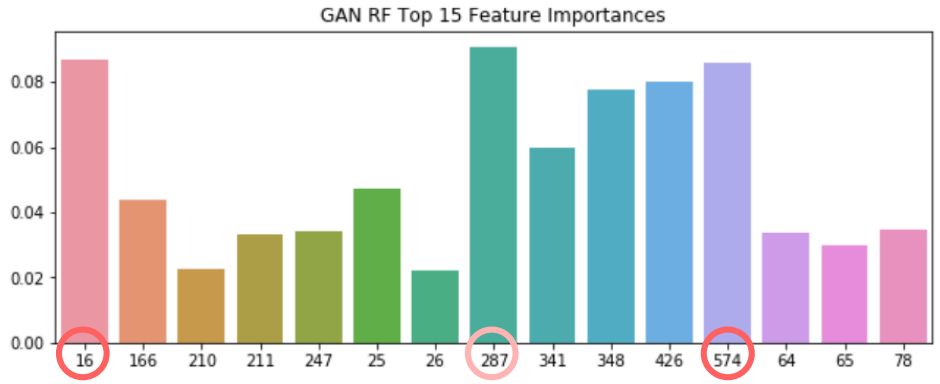
모델 선택 및 센서 시각화

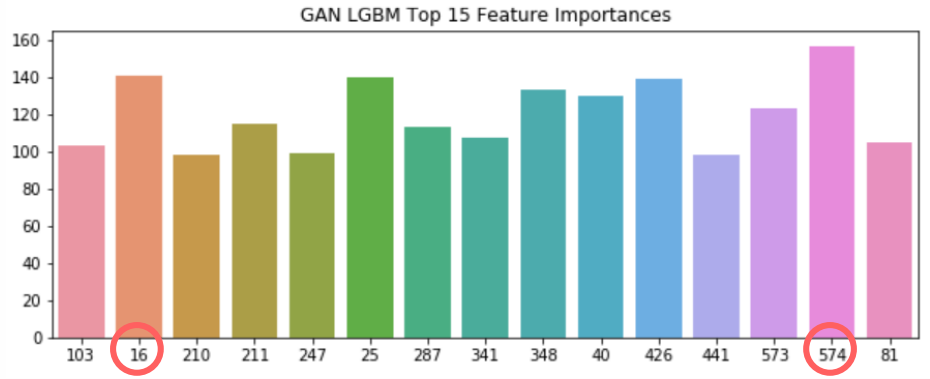
- 랜덤 포레스트 (93.63%)
- LGBM (93.63%)



GAN 이후에서는 동일하게 16, 574번 센서가 중요

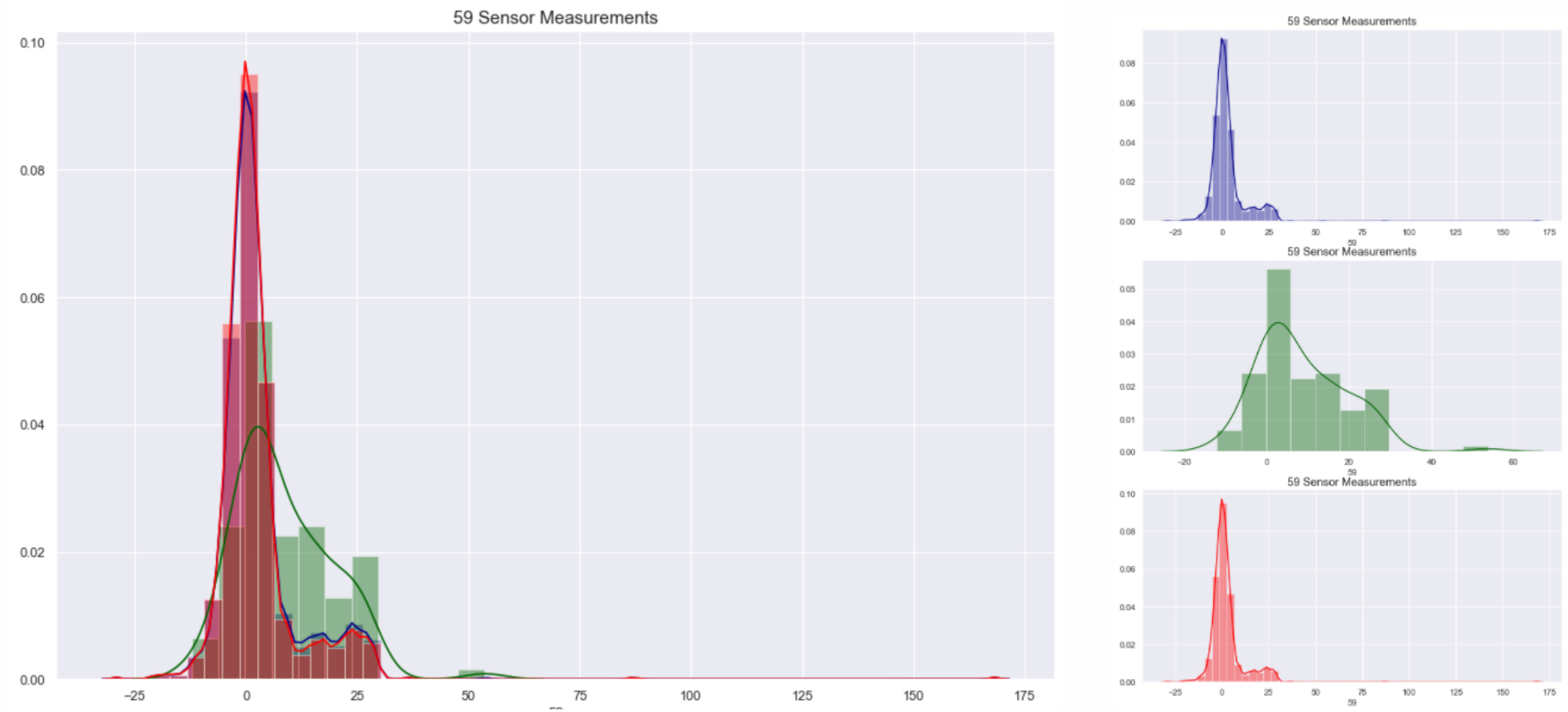
랜덤 포레스트는 287번 센서 역시 중요함





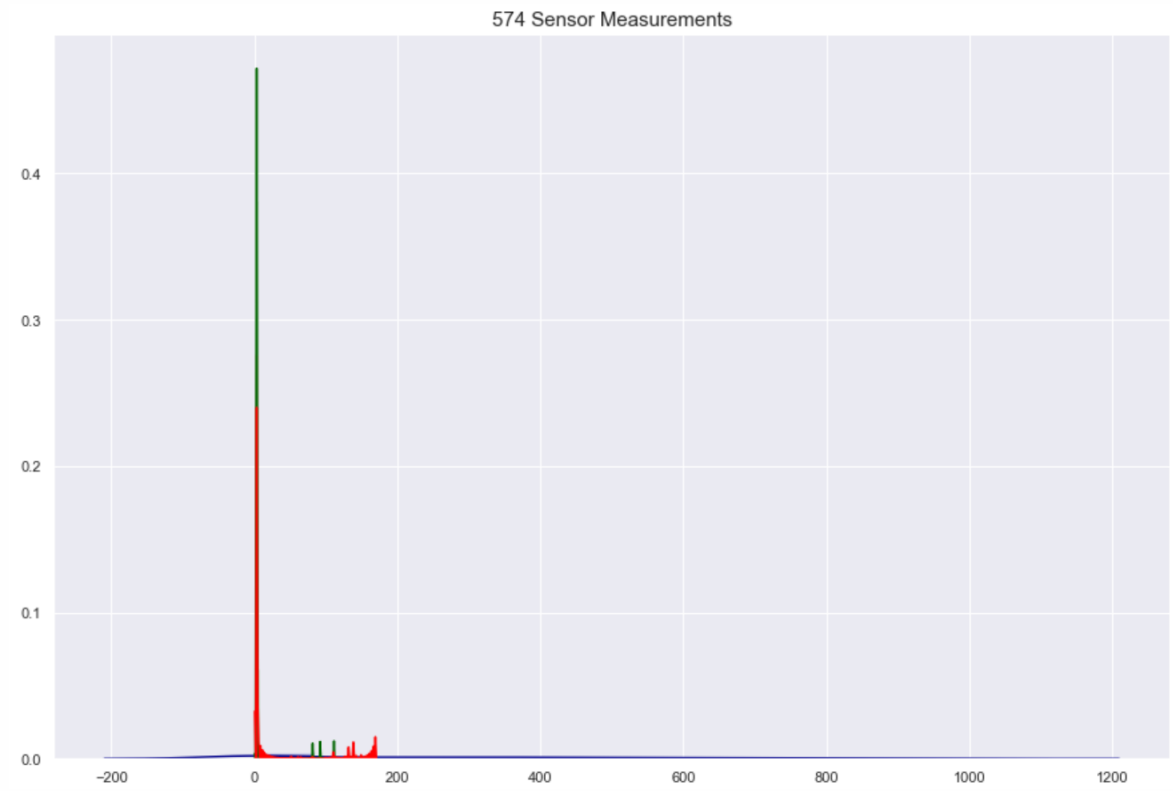
모델 선택 및 센서 시각화

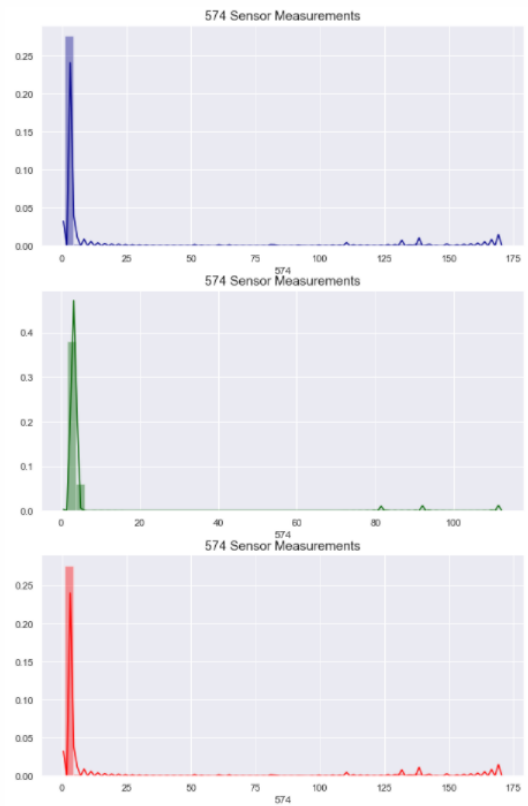
➢ 59번 센서



모델 선택 및 센서 시각화

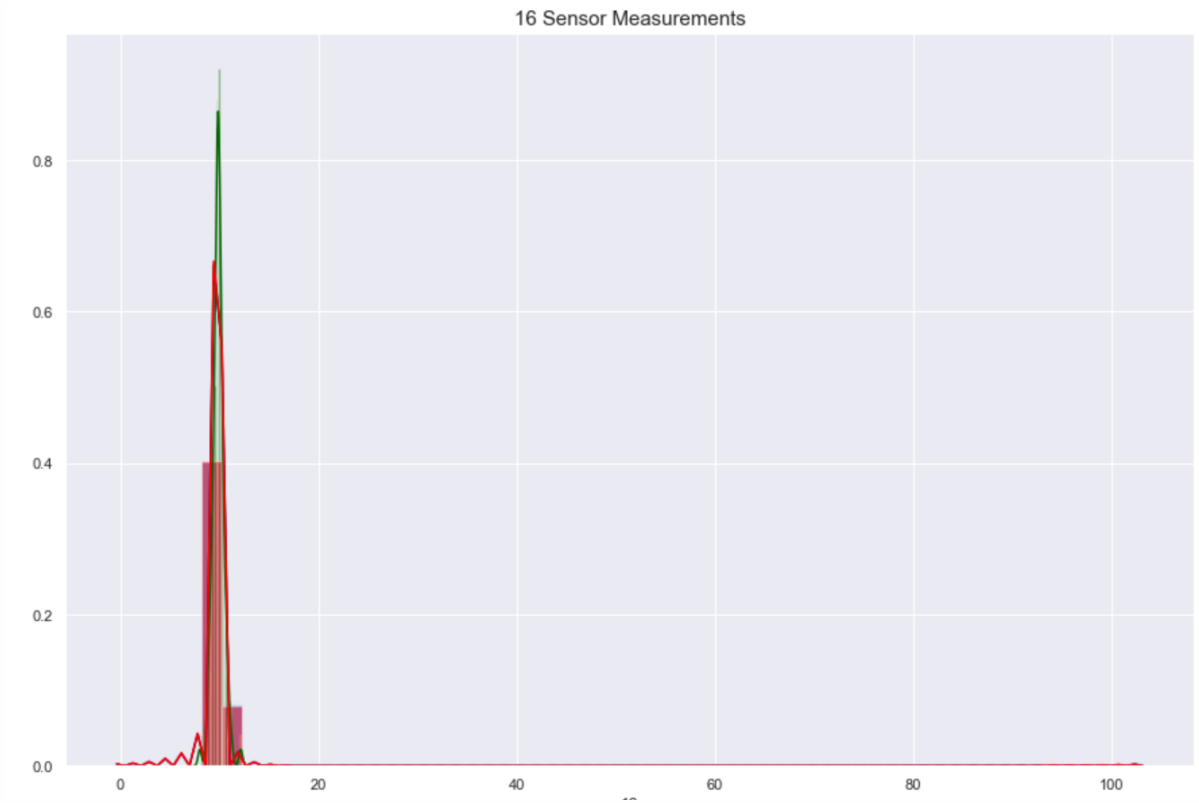
➢ 574번 센서

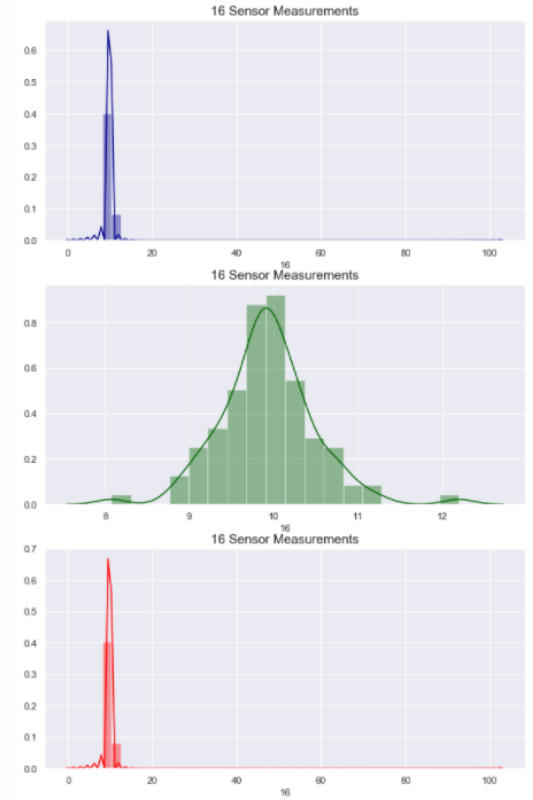




모델 선택 및 센서 시각화

➢ 16번 센서





감사합니다

Thank you