

# KAMP 제조AI데이터셋 활용 권한

No. GB20240909-PU1724919073-136318



## △ 제조AI데이터셋 활용 주의사항

연구/공식적으로 사용하실 때에는 꼭 아래와 같이 'KAMP 출처(reference)'를 남겨주시고, 인용 시 활용한 내용과 문서 등은 아래 이메일로 보내주시기를 바랍니다.  
(E-mail : kamp@kaist.ac.kr)

### 출처 표기 양식

중소벤처기업부, Korea AI Manufacturing Platform(KAMP), 교반구동장치 AI 데이터셋,  
KAIST((주)임피스, 한양대학교 산학협력단, (주)아큐라소프트), 2020.12.14., [www.kamp-ai.kr](#)

## 제조AI데이터셋 개요

제조AI데이터셋명 : 교반구동장치 AI 데이터셋

업종 : 식품가공

목적 : 예지보전

유형 : CSV

알고리즘 : K-최근접 이웃(K-NN) 회귀를 이용하여 표준데이터를 추출하고 <BR>이에 대한 불량 예측값과 실제 값을 비교하여 정확도를 확인한다.

사용조건 : 콘텐츠 변경허용

제공기관 : KAIST (수행기관: (주) 임피스/한양대학교 산학협력단/(주)아큐라소프트)

등록일 : 2020-12-14

## 이용자 정보

회원 ID : whdals0512

다운로드일 : 2024-09-09 15:35:47

활용목적 : 제조AI 활용 참조모델 확인

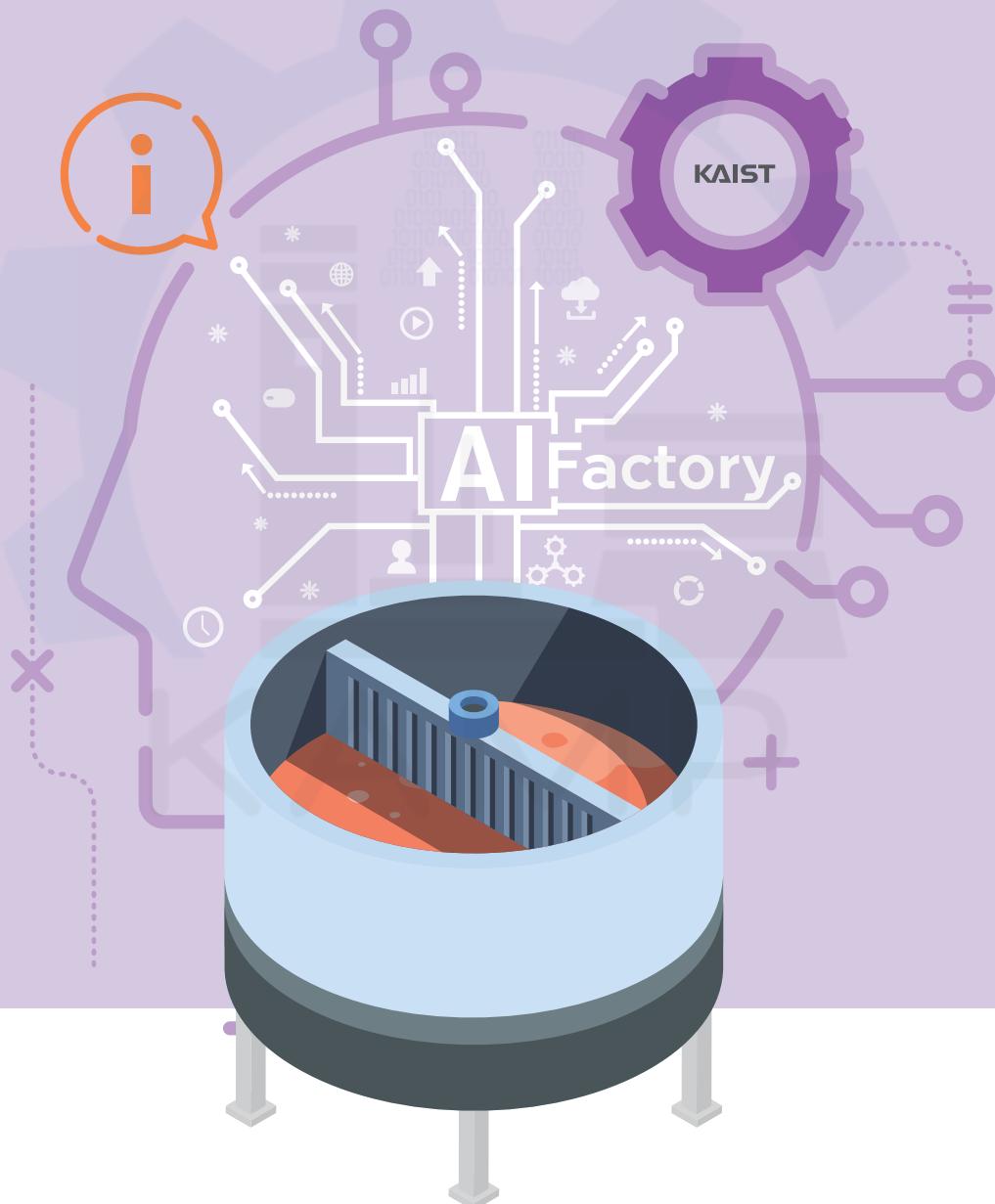
인공지능 중소벤처 제조 플랫폼(KAMP, Korea AI Manufacturing Platform)에서 다운로드한 제조AI데이터셋의 저작권을 위하여 문서 고유 번호와 워터마크를 부여하여, 해당 문서의 표준을 준수하였음을 증명합니다.

운영기관 : KAIST 제조AI빅데이터센터

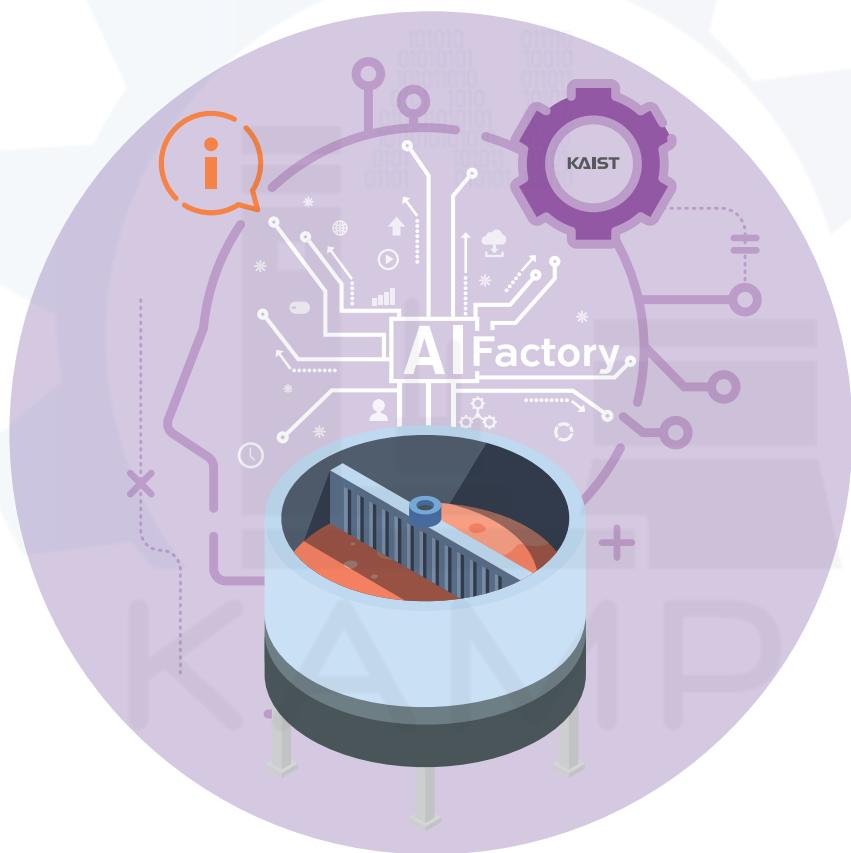
주소 : 대전광역시 유성구 문지로 193 KAIST 문지캠퍼스 행정동

전화번호 : 042-350-1331 | 이메일 : [kamp@kaist.ac.kr](#)

# 「교반구동장치 AI 데이터셋」 분석실습 가이드북

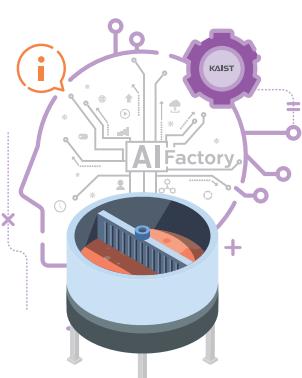


# 「교반구동장치 AI 데이터셋」 분석실습 가이드북



# Contents

<b>1 분석요약</b>	<b>04</b>
<b>2 분석 실습</b>	
<b>1. 분석 개요</b>	<b>05</b>
<b>1.1 분석 배경</b>	<b>05</b>
1) 공정(설비) 개요	
2) 이슈사항(Pain point)	
<b>1.2 분석 목표</b>	<b>06</b>
1) 분석목표	
2) 제조 데이터 정의 및 소개	
3) 제조 데이터 분석 기대효과	
4) 시사점(implication) 요약기술	
<b>2. 분석실습</b>	<b>08</b>
<b>2.1 제조데이터 소개</b>	<b>08</b>
1) 데이터 수집 방법	
2) 데이터 유형/구조	
<b>2.2 분석 모델 소개</b>	<b>10</b>
1) AI 분석모델	
<b>2.3 분석 체험</b>	<b>23</b>
1) 분석 단계별 Process	
2) 분석 실습	
[단계 ①] 데이터 불러오기	
[단계 ②] 데이터 기술통계	
[단계 ③] 데이터 집단 구분	
[단계 ④] 훈련용, 시험용 데이터 구분	
[단계 ⑤] 하이퍼 파라미터 탐색	
[단계 ⑥] 모델 학습	
[단계 ⑦] 모델 평가	
[단계 ⑧] 모델 평가 결과 시각화	
[단계 ⑨] 결과 분석 및 해석	
<b>3. 유사 타 현장의 「교반구동장치 AI 데이터셋」 분석 적용</b>	<b>51</b>
<b>부 록</b>	
<b>분석환경 구축을 위한 설치 가이드</b>	<b>52</b>



# 「교반구동장치 AI 데이터셋」 분석실습 가이드북

- 필요 SW : Jupyter Notebook, Anaconda 등
- 필요 패키지 : Numpy, Pandas, Matplotlib, Keras, Sklearn, Tensorflow, Ipython, Pydotplus, Os, Graphviz
- 분석 환경 : [CPU] i5-10400, [GPU] gtx 1660 super, [RAM] 32GB
- 필요 데이터 : mixing\_actuator.csv
- 주 관 기관 : 한국과학기술원(KAIST)
- 수행 기관 : (주)임픽스, 한양대학교 산학협력단, 주식회사 아큐라소프트



## 1 분석요약

No	구분	내용
1	분석 목적 (현장 이슈, 목적)	<ul style="list-style-type: none"> <li>- 교반구동장치가 부착된 용해탱크(용해공정)는 전처리 공정의 첫 번째 단계로, 분말 원재료를 정제 수 등에 녹이는 작업을 한다. 이같이 용해/혼합된 분말 및 액상 원재료는 후공정 단계에서 다시 분말화되기 때문에 본 공정에서 원재료가 균일하게 혼합되는 것이 매우 중요하다. 따라서 설비가 SOP에 따른 셋팅값대로 정상 작동하여 항상 동일한/균일한 품질을 생산하는 것이 중요해진다.</li> <li>- 특히, 용해탱크는 전 라인에 사용되는 설비이기 때문에 타 설비보다 설비의 사용시간 및 빈도가 높아 설비 교반구동장치의 예지보전 분석을 통한 고장관리 및 수명관리에 대한 니즈가 있다.</li> </ul>
2	데이터셋 형태 및 수집방법	<ul style="list-style-type: none"> <li>- 분석에 사용된 변수 : z축 진동데이터, 정상/결함 데이터</li> <li>- 데이터 수집 방법 : 무선 진동센서(구동기 부착)</li> <li>- 데이터셋 파일 확장자 : csv</li> </ul>
3	데이터 개수 데이터셋 총량	<ul style="list-style-type: none"> <li>- 데이터 개수 : 351,232개</li> <li>- 데이터셋 총량 : 19.3MB</li> </ul>
4	분석적용 알고리즘	<p>알고리즘</p> <p>"K-최근접 이웃(K-NN) 회귀를 이용하여 표준데이터를 추출하고 이에 대한 불량 예측값과 실제값을 비교하여 정확도를 확인한다."</p>
		<p>알고리즘 간략소개</p> <ul style="list-style-type: none"> <li>- K-NN 회귀는 목표값이 연속형이라는 점에서 K-NN 분류와 차이가 있다.</li> <li>- 특정 시점의 데이터에 대해 가장 가까운 K개 이웃 데이터의 평균이 예측값으로 계산되어 최종적인 예측선을 만든다.</li> <li>- 센서데이터의 패턴을 바탕으로 한 시계열적인 표준데이터를 임계치 개념으로 추출하고 표준편차를 이용하여 최소, 최대 임계치를 계산할 수 있다.</li> </ul>
5	분석결과 및 시사점	<ul style="list-style-type: none"> <li>- 설비 가동 중 모터의 진동데이터를 이용하여 결함 및 고장 징후를 검출할 수 있는 모델을 도출한다.</li> <li>- 용해탱크뿐만 아니라 구동장치가 있는 다른 설비에도 적용할 수 있는 모터 베어링 진동데이터의 수집 및 AI 모델 개발·검증을 통해 열악한 중소기업에 빅데이터 및 AI 기술을 적용하여, 실질적인 설비 관리 개선 및 생산성 향상에 기여한다는 점에서 시사하는 바가 크다고 판단된다.</li> </ul>

## 2 분석 실습

### 1. 분석 개요

#### 1.1 분석 배경

##### 1) 공정(설비) 개요

###### • 공정(설비) 정의 및 특징

- 용해탱크의 교반구동장치는 분말 원재료를 액상 원재료에 녹이는 과정에서 원료를 교반하여(섞어서) 더 잘 용해되도록 한다.
- 용해탱크의 교반구동장치는 교반(혼합/믹싱) 기능을 하는 부분으로, 크게 모터(motor), 감속장치(reducer), 회전축(mixing shaft), 교반용 날개(impeller) 등의 구조로 이루어진다.
- 모터는 교반기 구동을 위한 동력을 공급하고, 감속장치는 실제 교반에 필요한 RPM을 얻기 위해 감속을 하며, 회전축은 모터 동력으로 회전하여 임펠러에 회전력을 전달하고, 임펠러는 회전축으로부터 전달받은 회전력으로 탱크 안의 내용물을 휘저어 섞는 기능을 한다.

##### 2) 이슈사항(Pain point)

###### • 공정(설비)상의 문제현황

- 용해공정은 원재료의 전처리를 수행하는 첫 번째 단계이니만큼 본 공정의 품질이 후공정 및 완제품의 품질에 미치는 영향이 크다. 특히 원료 용해액을 후공정 단계에서 다시 분말화하기 때문에 전처리 단계에서 모든 원료가 균일하게 혼합되는 것이 매우 중요하다. 이러한 용해공정의 품질을 보장하기 위해서는 10여 종류를 넘는 분말 및 액상 원료를 섞는 교반기가 SOP에 따른 셋팅값대로 정상 작동하여 균일한 혼합액을 생산하여야 한다.
- 게다가 용해탱크는 전 제품 생산라인에 사용되는 설비이기 때문에 타 설비보다 사용 시간과 빈도가 높아 설비 교반구동장치의 고장관리 및 수명관리가 더욱 중요하다. 설비 결함 및 고장은 모터, 감속장치, 회전축, 날개 등 구조물이나 부품 중 어디에서나 발생할 수 있는데, 이 중 주요 부품인 모터 베어링은 전기 침식, 부적절한 윤활 및 오염, 진동으로 인한 손상, 부적절한 설치와 손상 설치, 불충분한 베어링 하중 등의 원인으로 인해 주로 고장이 발생한다.
- 모터 베어링의 이상으로 교반구동장치의 기능이 저하되거나 작동을 멈추면 바로 제품 품질 또는 생산 자체에 영향을 미치기 때문에, 첫 이상징후를 보일 때 대처하는 것

이 가장 좋으나 일반적으로 눈에 보이는 것이 아니므로 고장이 발생하는 한계점을 넘기 전에 이상징후를 발견하기가 쉽지 않다. 모터 베어링에 이상이 있을 경우 평상시와는 다른 소음과 진동이 발생하게 되는데, 이를 조기에 발견하기 위해 현장에서 종종 취하는 방법 중 하나는 청진기를 설비에 갖다 대고 구동부의 소리를 듣고 이상 여부를 찾아내는 것이다. 이같은 방법은 설명할 필요도 없이 고도의 능력과 오랜 경험과 노하우, 직감을 필요로 하며, 이마저도 인력에 공백이 생기는 경우 대처가 어렵다.

### • 문제해결 장애요인

- 공장마다 생산 사이클과 사용빈도가 다르고, 설비 운영방식과 생산제품이 다르기 때문에 설비와 부품은 각 업체에 맞게 관리가 필요하다. 그러나 가동상태나 제품품질의 변화를 통해 간접적으로 확인하는 시점에는 이미 늦은 상황이고, 직접적 확인은 해당 부위가 설비 내부에 위치하여 확인이 어렵거나, 육안으로 확인이 가능하여도 관련 전문가가 아닌 이상 판단할 수 없기 때문에 생산에 차질이 발생할 때까지 발견 못 하는 경우가 많다.

### • 극복방안

- 상기 조건 속에서 교반구동장치의 기능과 수명을 관리하고, 나아가 용해공정의 품질을 보장하기 위해서는 지금과 같이 아날로그 방식과 주관적 판단에 의존한 관리가 아닌 데이터에 기반한 예측을 통한 관리가 필요하다. 다양한 문제요인으로 인해 교반구동장치의 모터 베어링에 이상이 있을 때 발생하는 진동데이터와 정상작동 시에 발생하는 진동데이터의 패턴 분석을 통하여 이상징후를 예측하여 설비 수명관리 및 부품 교체관리에 이용할 수 있다.

## 1.2 분석 목표

### 1) 분석 목표

- 용해공정의 가장 중요한 작업 중 하나가 원료의 교반임에 따라 교반기에 이상이 있을 경우, 품질 및 생산 저하를 가져온다. 교반구동장치 주요 부품 중 하나인 모터 베어링은 전기 침식, 부적절한 윤활 및 오염, 진동으로 인한 손상, 부적절한 설치와 손상 설치, 불충분한 베어링 하중 등에 의해 파손/고장이 발생할 수 있으며, 이로 인해 비정상적 소음 또는 진동이 발생할 수 있다. 시간의 흐름에 따른 교반구동기 진동데이터의 패턴을 분석하여 이상치를 추출함으로써 교반기의 이상을 예측하고, 이를 기반으로 설비 예지보전을 수행하고자 한다.

## 2) 제조 데이터 정의 및 소개

- 교반구동장치의 모터는 베어링에 이상이 있으면 정상 작동시와는 다른 패턴의 진동이 발생하기 때문에, 구동부에 설치된 진동센서를 통해 수집한 정상 작동시의 z축 진동값과 비정상 작동시의 z축 진동값의 패턴을 분석하기 위한 데이터이다.

## 3) 제조 데이터 분석 기대효과

- 구동부 진동데이터 패턴에 따라 설비의 건강상태를 예측할 수 있는 설비에 넓게 활용할 수 있을 것으로 판단된다. 본 분석을 통해 정상작동 시와 이상작동 시의 모터 진동상태의 패턴을 도출하여, 예지보전을 통한 설비관리를 할 수 있게 하여, 공정 생산성과 품질을 보장할 수 있는 설비관리 최적화를 위한 분석 데이터셋을 구축한다.
- 특히, 고장이 나면 수리하거나 교체하는 사후보전이나 정기적으로 수리하는 계획보전이 아닌 지속적으로 측정되는 과학적 설비상태자료(진동, 온도 등)의 분석 후 필요 설비만 수리하는 예지보전(PdM: Predictive Maintenance)을 가능케 하여 효율적이고 효과적으로 설비관리를 할 수 있을 것으로 판단된다. 주관적 판단이 아닌 철저히 데이터를 기반으로 설비의 건강상태를 파악하고, 설비의 결함을 조기에 검출한다.

## 4) 시사점(implication) 요약기술

- 식품제조업체 H사 공장에서는 용해탱크 교반구동장치의 건강상태를 파악하기 위해 청진기를 통해 이상소음을 찾아내는 등 고도의 능력과 주관적 판단에 의존하여 그동안 효율적이고 효과적인 설비관리가 어려웠다. 따라서, 교반구동장치 모터의 결함여부를 정상작동과 이상작동 시의 진동데이터 패턴을 통해 판별 및 예측할 수 있는 AI 기법을 적용하여, 설비 가동중 모터의 진동데이터를 이용하여 결함 및 고장 징후를 검출할 수 있는 모델을 도출하고자 한다. 교반구동장치 모터 베어링의 진동데이터를 예제로 기본적인 데이터 탐색을 통해 전체 데이터 및 각 변수의 특성을 파악하는 방법을 학습하고, 이를 바탕으로 데이터 패턴 모델을 구성하고자 한다.
- 예제의 용해탱크뿐만 아니라 구동장치가 있는 다른 설비에도 적용할 수 있는 모터 베어링 진동데이터의 수집 및 AI 모델 개발·검증을 통해 열악한 중소기업에 빅데이터 및 AI 기술을 적용하여, 실질적인 설비관리 개선 및 생산성 향상에 기여한다는 점에서 시사하는 바가 크다고 판단된다.

## 2. 분석실습

### 2.1 제조데이터 소개

#### 1) 데이터 수집 방법

- 제조 분야 : 분무건조공법을 이용한 분말유크림 제조
- 제조 공정명 : 용해혼합(교반구동장치)
- 수집장비 : 무선 진동센서
- 수집기간 : 2020년 11월 18일 ~ 2020년 11월 20일 (약 3일)
- 수집주기 : 사이클타임 약 10분

#### 2) 데이터 유형/구조

NUM	Date	Sensor	Quality
0	2020-11-18T20:32:20.257000+00:00	0.14609375	PASSED
1	2020-11-18T20:32:20.257000+00:00	0.54453125	PASSED
2	2020-11-18T20:32:20.257000+00:00	1.38125	PASSED
3	2020-11-18T20:32:20.257000+00:00	3.1875	PASSED
4	2020-11-18T20:32:20.257000+00:00	4.47578125	PASSED
5	2020-11-18T20:32:20.257000+00:00	4.25	PASSED
6	2020-11-18T20:32:20.257000+00:00	3.17421875	PASSED
7	2020-11-18T20:32:20.257000+00:00	1.81953125	PASSED
8	2020-11-18T20:32:20.257000+00:00	-0.8234375	PASSED
9	2020-11-18T20:32:20.257000+00:00	-1.95234375	PASSED
10	2020-11-18T20:32:20.257000+00:00	-4.36953125	PASSED
11	2020-11-18T20:32:20.257000+00:00	-6.5609375	FAILED
12	2020-11-18T20:32:20.257000+00:00	-6.83984375	FAILED
13	2020-11-18T20:32:20.257000+00:00	-5.03359375	FAILED
14	2020-11-18T20:32:20.257000+00:00	-2.90859375	PASSED
15	2020-11-18T20:32:20.257000+00:00	-3.75859375	PASSED
16	2020-11-18T20:32:20.257000+00:00	-1.95234375	PASSED
17	2020-11-18T20:32:20.257000+00:00	1.115625	PASSED
18	2020-11-18T20:32:20.257000+00:00	2.3640625	PASSED
19	2020-11-18T20:32:20.257000+00:00	-1.81953125	PASSED
20	2020-11-18T20:32:20.257000+00:00	-1.3984375	PASSED
21	2020-11-18T20:32:20.257000+00:00	1.296875	PASSED
22	2020-11-18T20:32:20.257000+00:00	3.1015625	PASSED

[그림 1] 교반구동장치 데이터 스크린샷

- 데이터 크기, 데이터 수량 : 4개 칼럼, 351,232개의 관측치

- 데이터 속성정의 표 :

변수명	설명	데이터 타입
Unnamed	자동 생성 인덱스	int64
Date	날짜, 시간(YYYY:MM:DD:hh:mm:ss)	object
Sensor	z-축 진동데이터	float64
Quality	결함여부	object

\* int64 = 정수, object = 문자, float64 = 실수

- 본 가이드북의 데이터셋은 데이터 인덱스(NUM), 데이터 수집일시(Date), z축 진동 데이터(Sensor), 구동부 결함여부(Quality) 등 4개 칼럼으로 이뤄져 있으며, 이 중 진동데이터(Sensor)만 연속형 수치 형태이다.
- 구동부 결함여부(Quality) 값 'PASSED'는 정상상태, 'FAILED'는 이상상태(결함/고장)를 의미한다.

- 기술통계 :

구분	Sensor
개 수	351,232
평균	-0.108910
표준편차	2.563446
최소값	-9.023438
중간값	-0.140625
최대값	9.203906
최빈값	-0.1484375

\* 실제 알고리즘에 사용하는 데이터 중 수치 데이터만 기술

- 독립변수/종속변수 정의 :

구분	명칭	비고
독립변수	하나의 패턴 내 시간	분석 코드에서 직접 생성
종속변수	z-축 진동데이터	

\* 독립변수란 다른 변수에 영향을 받지 않는 변수로, 입력값이나 원인을 나타낸다.

종속변수란 독립변수의 변화에 따라 어떻게 변하는지를 알고자 하는 변수를 말하며, 결과물이나 효과를 나타낸다.

## 2.2 분석 모델 소개

### 1) AI 분석모델

- 해당 AI 방법론(알고리즘) 선정 이유 기술

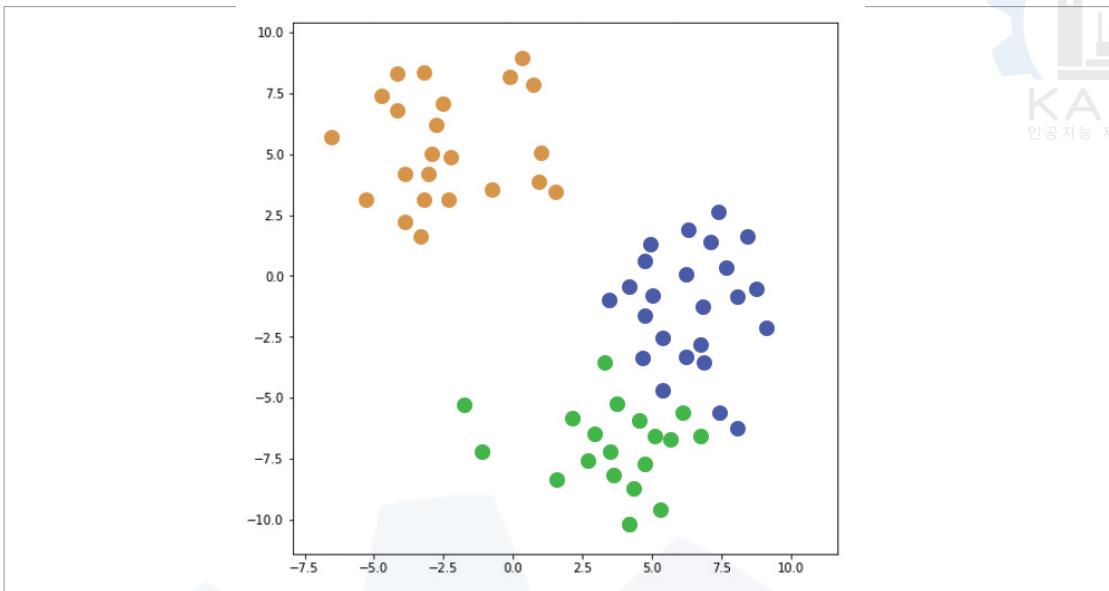
- 교반구동장치 데이터 분석 방법 : K-최근접 이웃 회귀

- K-최근접 이웃(K-Nearest Neighbor) 회귀를 이용하여 모터 베어링의 이상징후 예측이 가능한 표준데이터를 추출할 수 있다. 특히 표준데이터를 시간의 흐름에 따라 유연하게 적용 가능하도록 추출한다. 이를 이용하면 각 업체의 설비와 부품에 적합한 모니터링이 가능하다. 또한, 육안으로의 이상징후 확인은 이미 그러한 징후가 발생한 지 오랜 시간이 지난 시점일 수 있다는 문제점이 있으나 K-최근접 이웃 회귀는 사전에 표준데이터를 추출해놓는 원리이므로 그러한 문제점을 방지할 수 있다. 따라서 K-최근접 이웃 회귀를 통해 생산에 차질이 발생하기 전에 이상징후를 발견 할 가능성을 높이고자 하였다.
- K-최근접 이웃 회귀 알고리즘은 분류와 회귀로 나뉘며 본 분석에서는 ‘회귀’를 중심으로 센서데이터 불량 검출 및 예지보전을 위한 표준데이터를 추출한다.
- K-최근접 이웃 회귀를 사용하는 가장 주된 이유는 센서데이터의 시계열적 패턴에 적합한 표준데이터(임계치)를 추출하기 위함이다. 임계치는 센서데이터의 수치를 통해 불량을 검출할 때 사용하는 기준이다. 이때 시구간 영역에 상관없이 모든 시점에 동일한 임계치를 적용할 경우, 시간에 따라 변화의 진폭이 달라지는 경우를 현실적으로 검출하기 어렵다는 문제점이 있다.
- 따라서 이러한 문제점을 해결하고자 수집된 데이터에 K-최근접 이웃 회귀 알고리즘을 적용하여 하나의 표준데이터 예측선을 추출한다. 즉, 하나의 시점에서 종속변수(센서데이터)와 가장 가까운 k개 이웃 데이터의 평균이 예측값으로 계산되어 이들로 이루어진 예측선을 최종적으로 도출한다. 이를 통해 예측선에 어긋나는 패턴이 검출되는 횟수와 예측이 얼마나 정확히 이루어졌는지 확인할 수 있다.

- 적용하고자 하는 AI 분석 방법론(알고리즘)의 구체적 소개

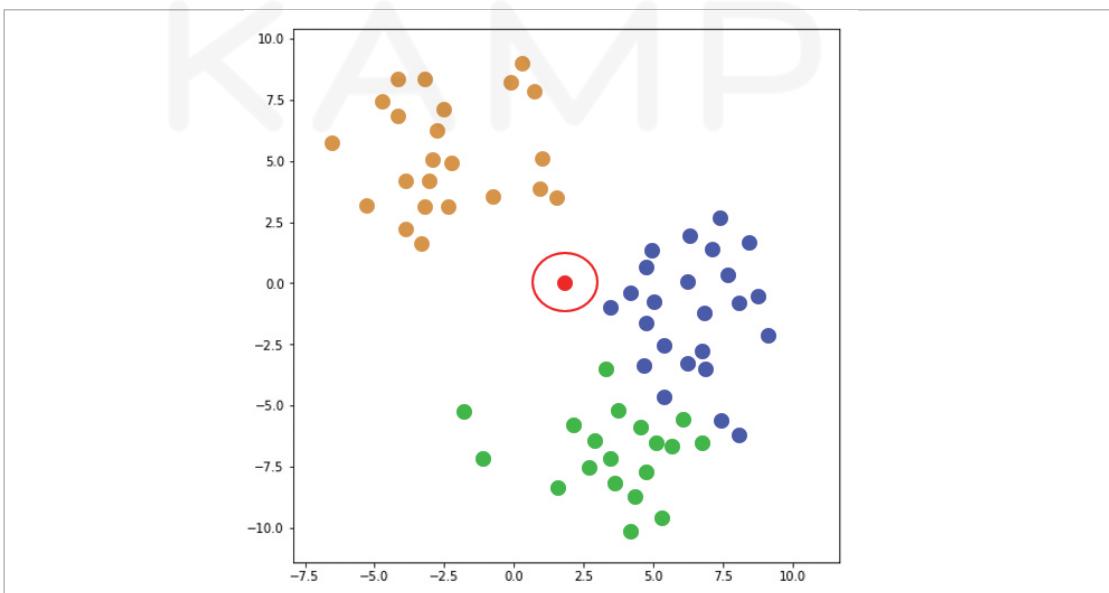
- K-최근접 이웃 회귀 관련 개념 ① : K-최근접 이웃 분류

- K-최근접 이웃 회귀의 구체적인 개념은 K-최근접 이웃 분류와 이에 회귀 방법론을 적용하는 과정을 통해 확인 가능하다.



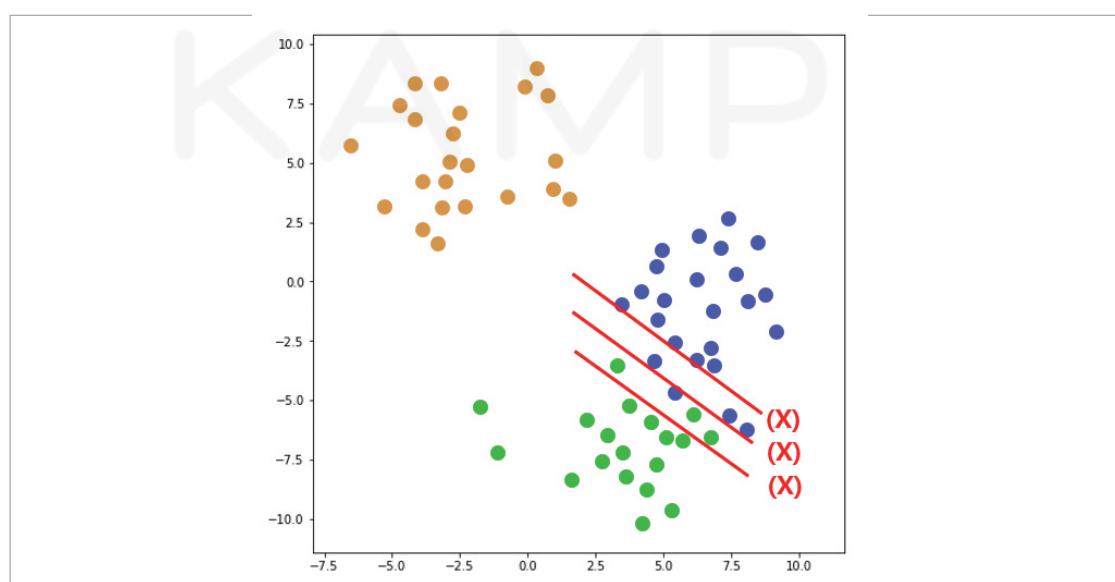
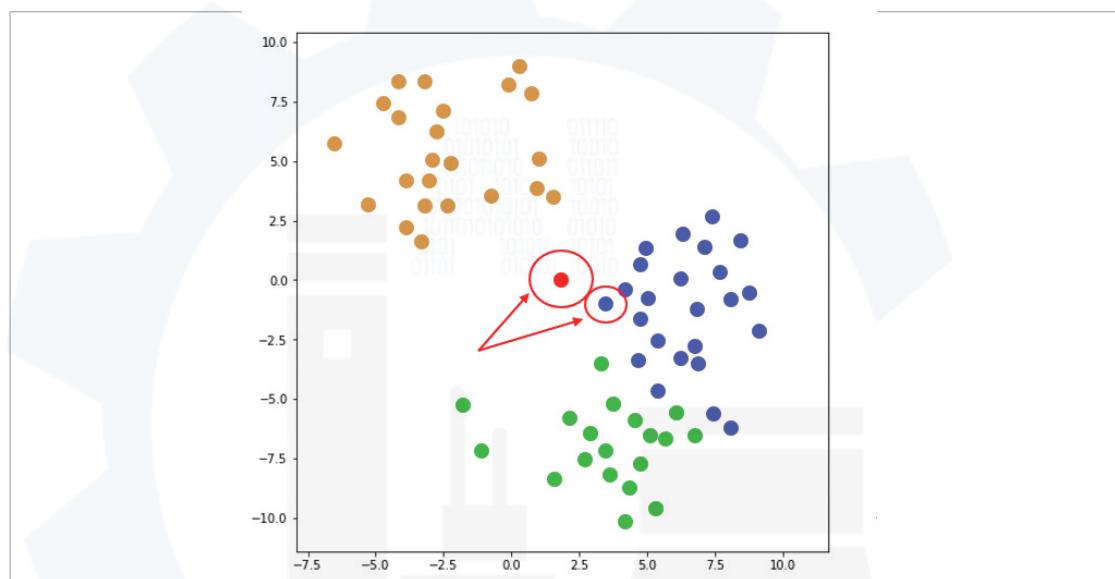
[그림 2] 2차원 그래프 상의 분류 데이터

- 먼저 K-최근접 이웃 분류 알고리즘의 개념을 소개하기 위한 간단한 데이터 분류 예시는 다음과 같다. 예시에서 사용될 데이터를 [그림 2]에 표시하였다. 구분된 색상에서 알 수 있듯이 데이터는 3가지 집단으로 분류된다. 설명의 용이성을 위해 파일을 분류한다고 가정하고 (1) 주황색 집단: 오렌지, (2) 파란색 집단: 포도, (3) 초록색 집단: 사과로 정의하였다.
- K-최근접 이웃 분류는 이러한 상황에서 종류를 알 수 없는 파일이 새롭게 주어졌을 때, 이 파일이 어떤 종류인지 분류하기 위해 사용한다. 이때, 새로운 데이터에 부여하게 될 종류는 ‘현재 갖고 있는 파일 종류 데이터’이다. 즉, 새로운 데이터가 종류를 직관적으로 판단하기 어려운 위치에 존재한다면, 이를 적절한 집단에 포함시키기 위해 K-최근접 이웃 분류 알고리즘을 사용한다.

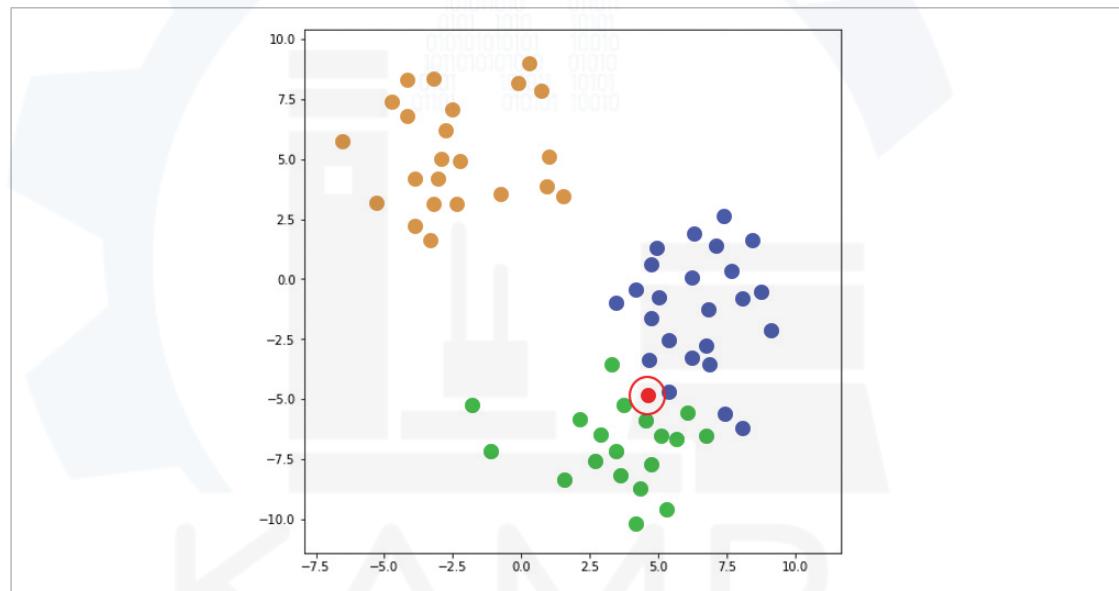


[그림 3] 분류 데이터에 추가된 새로운 데이터

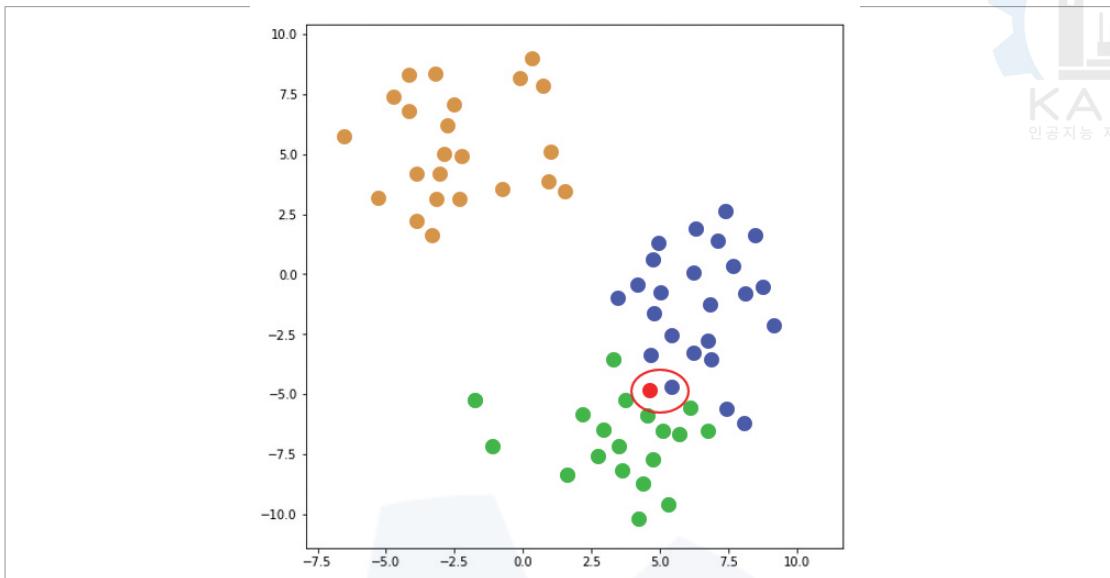
- 새로운 파일 데이터가 주어졌다고 가정하고, [그림 3]의 가운데에 새로운 데이터를 표시하였다. 이러한 상황에서 ‘새로운 파일은 오렌지와 포도 중 어떤 파일에 가장 가까울까?’라는 직관적인 질문을 고려할 수 있다. 그리고 2차원 그래프 상에서 새로운 데이터가 ‘거리상 가장 가까운’ 데이터와 유사한 집단에 속할 것이라고 예상 가능하다.
- K-최근접 이웃 분류에서는 이때의 ‘가장 가까운 데이터’를 ‘최근접 이웃(Nearest Neighbor)’으로 정의한다. 거리상 가장 가까운 데이터들이 모여 있는 집단을 육안으로 식별하는 것은 현실적으로 불가능하다. 따라서 새로운 데이터와 기존 데이터들의 거리 계산을 위해 K-최근접 이웃 분류 알고리즘은 특정 ‘거리 계산식’을 이용한다. 예를 들어, K를 1로 설정한다면 이는 ‘가장 가까운 1개( $K=1$ )의 이웃이 어떤 집단에 속하는지 확인하고 새로운 데이터도 같은 집단으로 간주하라’라는 의미이다.



- [그림 4]에 표시된 바와 같이, 거리상 파란색 데이터가 새로운 데이터와 가장 가까우므로 ‘새로운 데이터는 포도이다.’라는 결론을 내릴 수 있다.
- 즉, K는 K-최근접 이웃 분류 알고리즘에서 새로운 데이터를 분류해내는데 가장 중요한 파라미터이다. 앞선 예제에서는 K를 1로 가정하였지만, 실제 분석에서는 K를 2 이상으로 설정하는 것이 일반적이다. 그 이유를 위의 [그림 5]를 통해 확인할 수 있다. 파란색과 초록색 집단을 보면 집단 간의 경계가 모호하다. 즉, 두 집단이 가깝게 맞닿아있는 부분으로 인해 경계를 정확히 정의하기 어렵다. 따라서 K가 1일 경우, 가장 가까운 데이터로 새로운 데이터를 분류하는 것은 사실 분류 정확도가 낮은 경우가 많다. 과일이 아니더라도 현실의 여러 예시를 생각해보았을 때 모든 집단이 명확히 구분되며 그 경계가 뚜렷한 경우는 드물다. 이러한 특징을 ‘데이터의 노이즈 (noise)’라고 한다. 경계가 뚜렷하지 않은 특징을 정확한 분류에 대한 ‘일정한 잡음’이라고 간주하는 것이다. 분석 알고리즘은 현실에서 활용하기 위한 것이므로 이러한 노이즈를 필수적으로 고려하고 파라미터를 설정해야 한다.

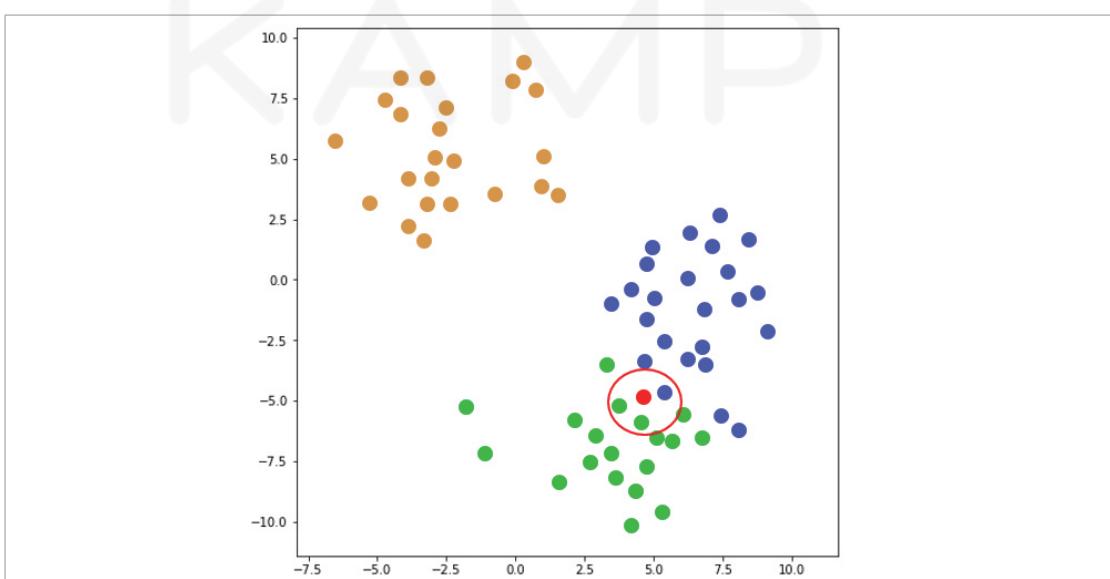


[그림 6] 분류된 집단의 경계에 새롭게 추가된 데이터

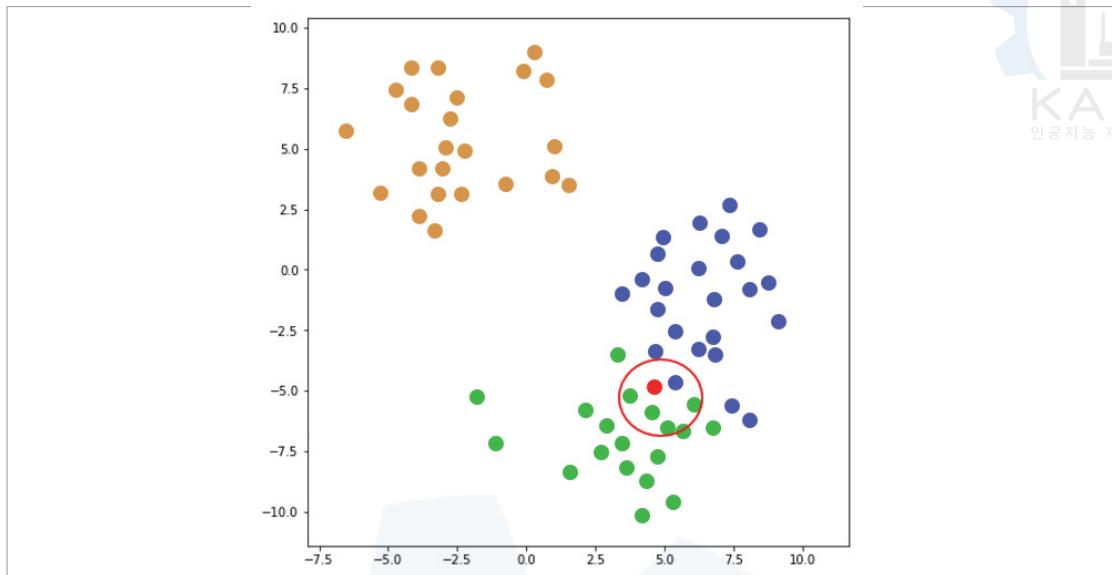


[그림 7] 집단 간 경계에서 새로운 데이터와 가장 근접한 파란색 데이터

- 더욱 현실적인 새로운 데이터가 투입됐다고 가정하고 이를 [그림 6]에 표시하였다. 데이터는 파란색과 초록색 집단 사이에 위치해 있다. [그림 7]을 보면 데이터는 육안으로의 거리상 파란색 데이터와 가장 가까운 것을 알 수 있다.
- 그런데 여기서 주목할 점은, [그림 7]에서의 가까운 파란색 데이터가 집단 내에서 ‘noise’로 간주될 수 있는 위치에 존재한다는 것이다. 즉, 기존의 개별 데이터와 새로운 데이터의 거리로 봤을 때 위에 표시한 파란색 데이터가 가장 가까운 것이 맞지만, 집단을 기준으로 확인했을 때 그 데이터는 사실상 집단 내 잡음일 수 있다는 것이다. 이는 가까운 파란색 데이터가 위치상 초록색 집단에 더 가까울 수도 있음을 의미한다. 따라서 K를 1로 설정했을 때 이러한 상황에서 새로운 데이터가 파란색 데이터로 분류된다면 정확히 분류된 것이 맞는지에 대한 의문이 제기될 수 있다.



[그림 8] K가 3일 때 새로운 데이터와 가장 가까운 3개의 데이터

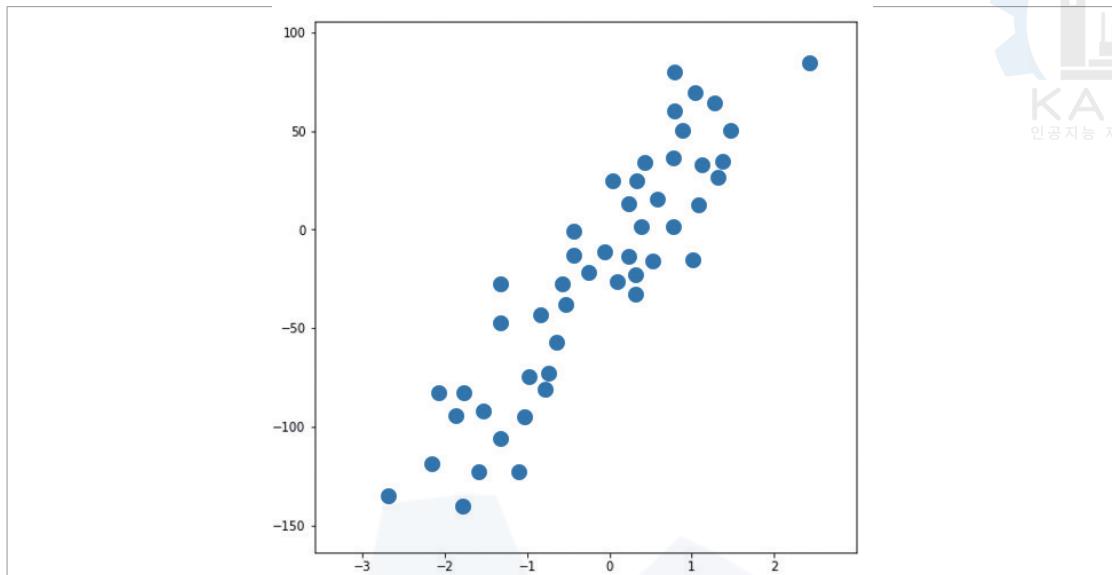


[그림 9] K가 5일 때 새로운 데이터와 가장 가까운 5개의 데이터

- 만약 이처럼 집단 간 경계가 모호한 상황을 감안하여 K를 3으로 설정하고 새로운 데이터와 가장 가까운 3개의 데이터를 확인하면 [그림 8]과 같다. 새로운 데이터 주변에 파란색 데이터 1개와 초록색 데이터 2개가 표시되었다. K를 5로 설정한다면, [그림 9]와 같이 새로운 데이터 주변에 파란색 데이터 1개와 초록색 데이터 4개가 표시된다. 따라서 K를 3과 5로 설정했을 경우, 1로 설정했을 때와 다르게 ‘새로운 데이터는 사과(초록색 데이터)이다.’라는 분석 결과를 산출할 수 있다.
- 이처럼 K-최근접 이웃 분류 알고리즘은 새로운 데이터와 가까운 점을 2개 이상 확인함으로써 데이터 집단 간 경계의 모호성이 야기할 수 있는 문제를 해결한다. 또한, K를 2 이상으로 설정하면 ‘모델 과적합’을 방지할 수 있다. 모델 과적합이란 분석 모델을 현재 주어진 데이터에 대해서만 완전히 최적화시키는 것을 의미한다. 따라서 과적합이 발생하면 기존 데이터를 훈련시킬 때의 예측력은 확실히 높지만 새로운 데이터가 들어왔을 때는 거의 예측이 되지 않는다. 쉽게 말해서, 과적합은 사람에게  $1+1=2$ 라는 사실에 대해서만 학습시킨 후  $100+1=2$  무엇인지 맞추라고 말하는 것과 같다. 따라서 K를 1보다 크게 설정하면 2차원 그래프 상 데이터의 절대적인 위치만을 고려하더라도 이러한 문제가 발생할 가능성이 감소한다.

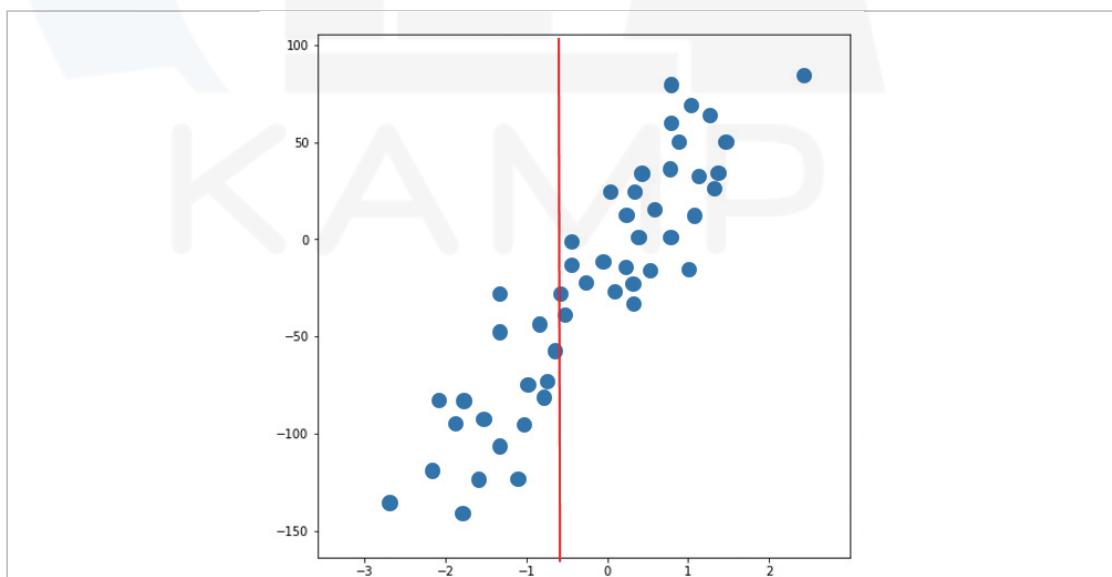
#### - K-최근접 이웃 회귀 관련 개념 ② : 회귀 방법론

- K-최근접 이웃 회귀는 K-최근접 이웃 분류 알고리즘과 유사하게 가까운 이웃 데이터들을 고려하지만 가까운 이웃 데이터들로 집단을 구분하는 것이 아니라 ‘개별값을 예측’한다는 차이가 있다.

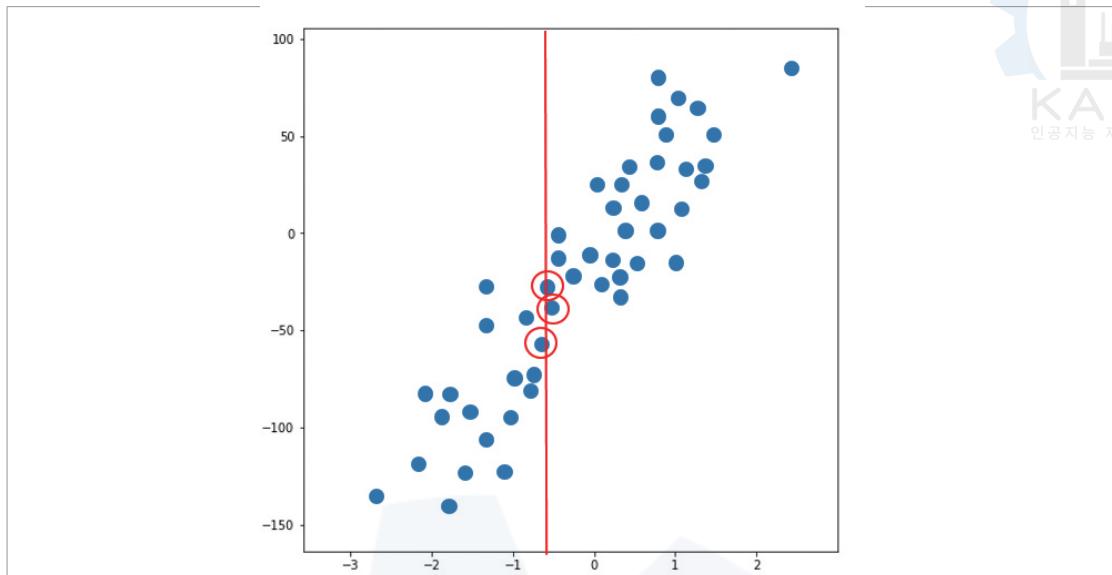


[그림 10] 2차원 그래프 상의 선형 데이터

- 이에 대한 설명을 위해 [그림 10]에 선형 데이터 예시를 표시하였다. 이 데이터는 개별값을 랜덤하게 생성하되 선형을 띠도록 하였으며 ‘선형 회귀분석’에서는 데이터가 이와 같이 필수적으로 선형이어야 한다.
- K-최근접 이웃 회귀를 이용하면 K개의 이웃 데이터를 통해 선형회귀분석을 이용했을 때와 유사한 ‘회귀선’을 도출할 수 있다. 선형회귀분석과 다른 점이 있다면 K-최근접 이웃 회귀의 분석 결과는 하나의 회귀식이 아니라 ‘주어진 입력을 바탕으로 가장 잘 예측된 평균값들의 집합’이라는 것이다. 즉, 종속변수에 대한 독립변수의 영향력을 의미하는 회귀계수가 별도로 확인되지 않으므로 혼동하지 않도록 주의해야 한다.

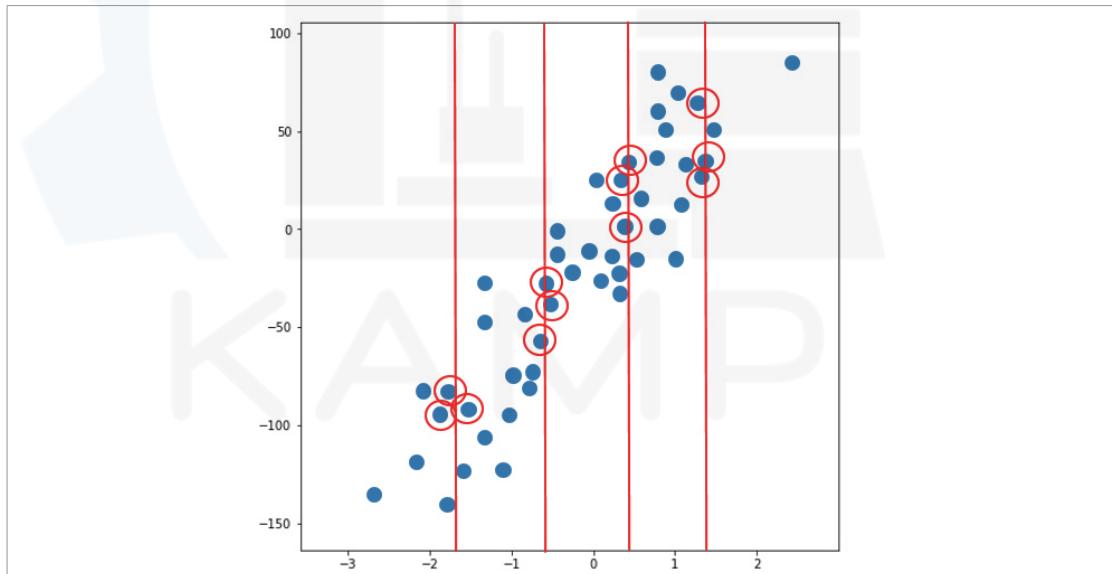


[그림 11] 선형 데이터에서  $X$ 값이  $-0.75$ 일 때의 수직선



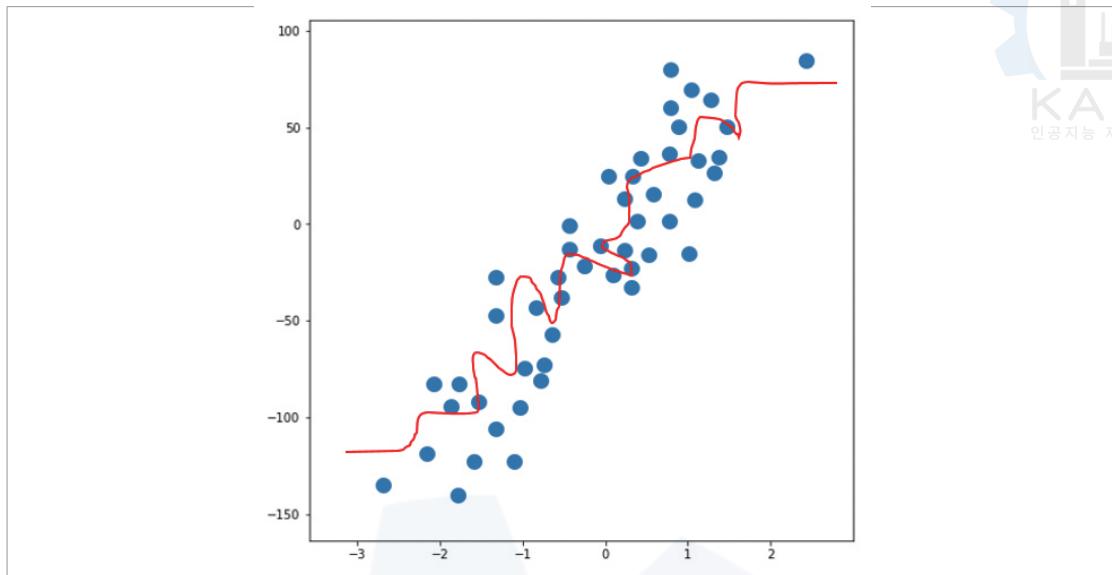
[그림 12] X값이 -0.75일 때의 수직선을 이용한 Y값 예측 방식

- [그림 11]에는 X값이 -0.75일 때의 수직선을 표시하였다. K-최근접 이웃 회귀는 수직선을 기준으로 -0.75 주변의 값들을 고려하여 출력될 예측값을 정한다. 즉, [그림 12]와 같이 X값이 -0.75인 지점에서 Y값을 예측하기 위해 해당 지점에서 수직선과 가장 가까운 K개의 데이터 값을 확인하고 이를 평균 내어 시점 당 하나의 예측값을 출력한다.



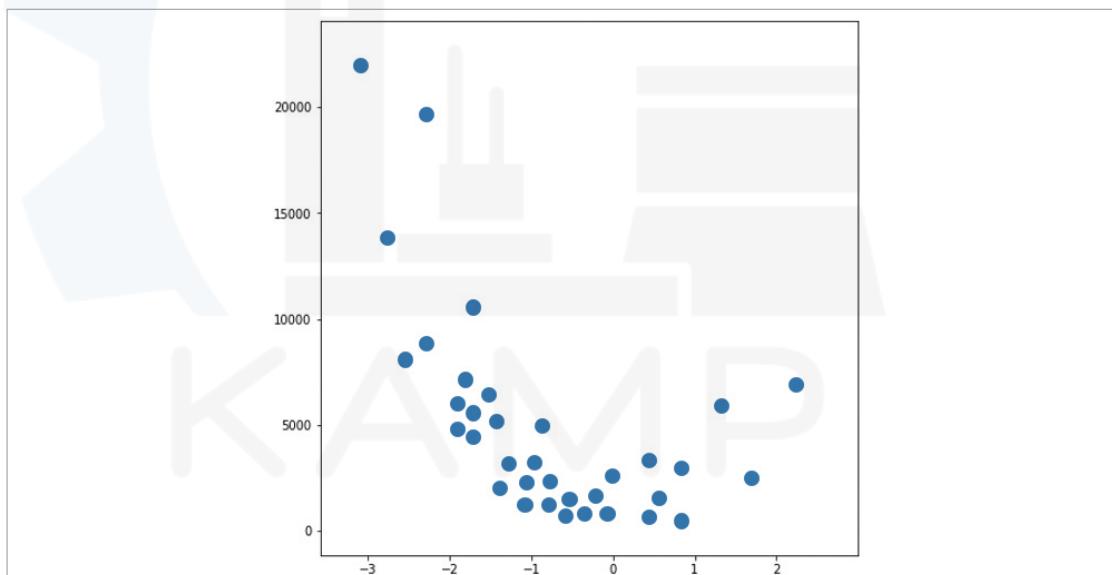
[그림 13] 각 시점에서의 Y값 예측 방식

- K-최근접 이웃 회귀는 데이터가 존재하는 모든 시점에 대해 위와 같은 방식의 계산을 수행한다. 예를 들어 X축이 ‘시간’이라면 모든 시점의 평균 데이터를 예측하게 되는 것이다. [그림 13]에는 간단한 예시로 4개의 수직선만 표현하였다.

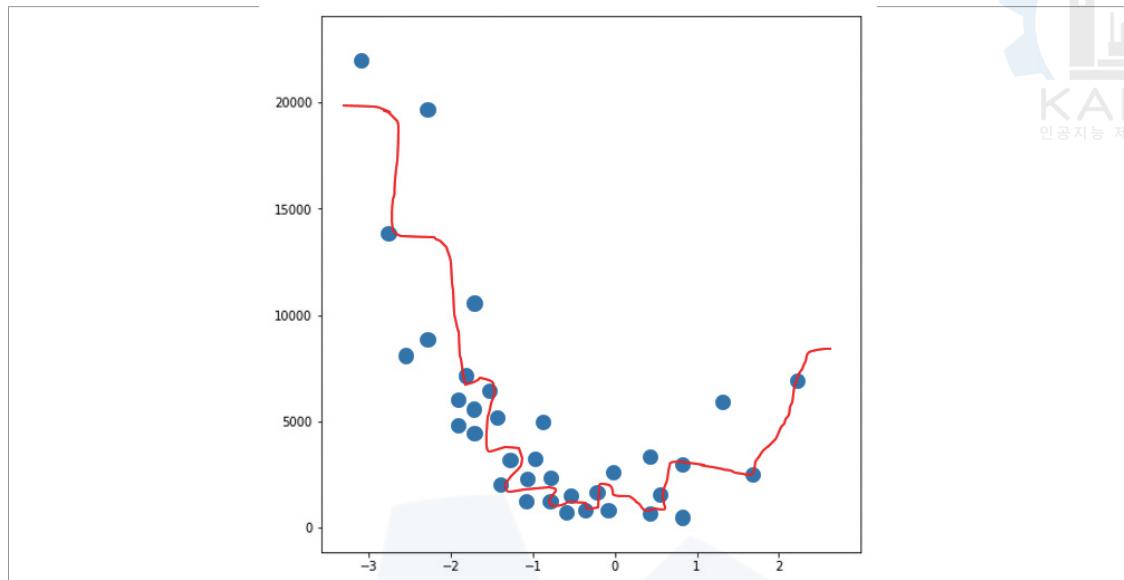


[그림 14] 선형 데이터에서 예측된 Y값을 연결한 하나의 예측선

- 모든 시점에서 예측된 값들을 하나의 선으로 연결하면 [그림 14]와 같이 예측에 사용된 데이터들에 대한 표준데이터를 표시할 수 있다. [그림 14]에서는 연결된 선이 데이터의 형태와 완벽히 일치하지 않지만, 일정한 상승 형태를 표현하고 있다고 할 수 있다.



[그림 15] 2차원 그래프 상의 비선형 데이터



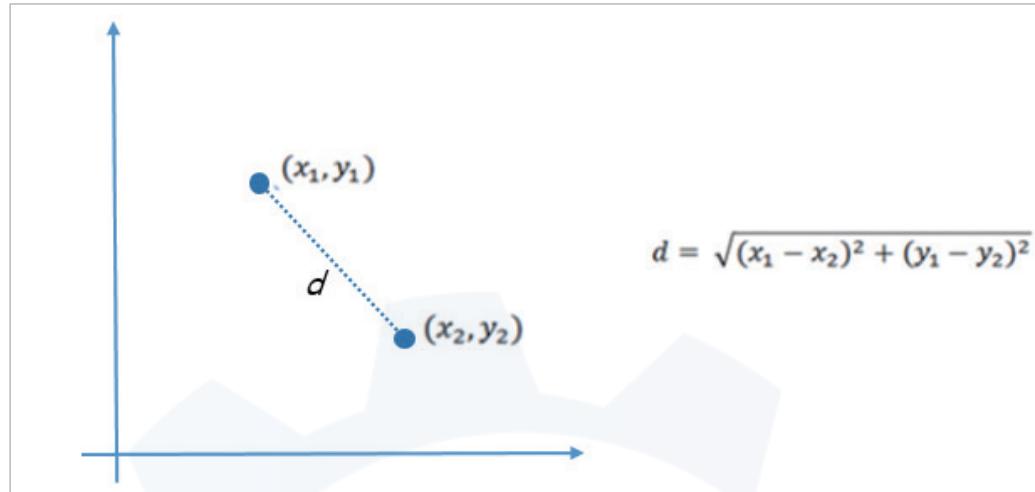
[그림 16] 비선형 데이터에서 예측된 Y값을 연결한 하나의 예측선

- K-최근접 이웃 회귀 알고리즘은 [그림 15]에 표현한 것처럼 비선형 데이터에도 적용 가능하다. 선형회귀분석과 다르게 데이터의 형태가 선형이 아니어도 분석이 가능하다는 의미이다. 데이터가 비선형을 뛴다는 것은 현실에서 상관 및 인과관계가 일정하게 성립되지 않고 발생하는 다양한 사건과 유사한 개념이다. 즉, K-최근접 이웃 회귀 알고리즘은 이러한 데이터의 형태에 구애받지 않으며 비교적 현실적인 문제에 대한 솔루션을 제시할 수 있다.
- 앞선 예시와 유사하게 비선형 데이터에 K-최근접 이웃 회귀 알고리즘을 적용한 결과는 [그림 16]과 같다. K를 2 이상으로 설정했을 때, 생성된 예측선이 데이터를 정확히 지나가진 않지만, 데이터의 흐름과 유사한 모양임을 알 수 있다. 즉, 과적합을 크게 걱정하지 않아도 되는 상태라고 할 수 있다.
- 이처럼 K-최근접 이웃 회귀는 K-최근접 이웃 분류의 개념에 선형회귀분석 개념이 추가된 알고리즘이다.

#### - K-최근접 이웃 회귀의 하이퍼 파라미터(Hyper parameter)

- 하이퍼 파라미터란 해당 알고리즘을 활용할 때 필수적으로 설정해야 하는 항목을 의미한다. K-최근접 이웃 회귀에서 중요한 파라미터는 ‘K(최근접 이웃 수)’, ‘거리 함수(데이터 간의 거리 계산식)’이다.
- K는 새로운 데이터와 가까운 이웃 데이터 수를 의미하며 실제로 분석에 활용할 때는 ‘K개의 이웃 데이터’라고 해석한다. K개의 집단으로 분류하는 것이 아닌, 개별 데이터 주변에 가까운 K개의 데이터를 찾아보는 것임을 유의해야 한다. 일반적으로 K 값은 1~20 범위 내에서 훈련 데이터 개수의 제곱근으로 설정한다. 하지만 이는 데이터에 따라 항상 상이하다. K를 선택함에 있어서 하나의 클러스터링 안에 클래스 수

가 그대로 나오는 경우를 피하기 위해 홀수를 사용하기도 한다. 가장 많이 사용되는 방법은 학습용 데이터와 테스트 데이터를 나누고 다양한 K값에 대해 제일 작은 예측 오차율이 산출되는 K를 선택하는 방법이다.



[그림 17] 유클리디안 거리식

- K-최근접 이웃 알고리즘은 기본적으로 새로운 데이터와 기존 데이터의 ‘거리’를 계산한다고 하였다. 이를 위한 대표적인 거리 함수는 ‘유클리디안 거리(Euclidean distance)’가 있다. 유클리디안 거리는 [그림 17]과 같은 식으로 계산되며 두 데이터 사이를 잇는 직선의 길이를 직접 구하는 것을 의미한다.

#### - K-최근접 이웃 회귀의 성능 평가

- 표준데이터를 이용하여 검출된 불량 여부와 실제 불량 여부의 일치성을 확인하기 위한 분류 성능 평가 척도로 컨퓨전 매트릭스(Confusion Matrix)를 이용한다.

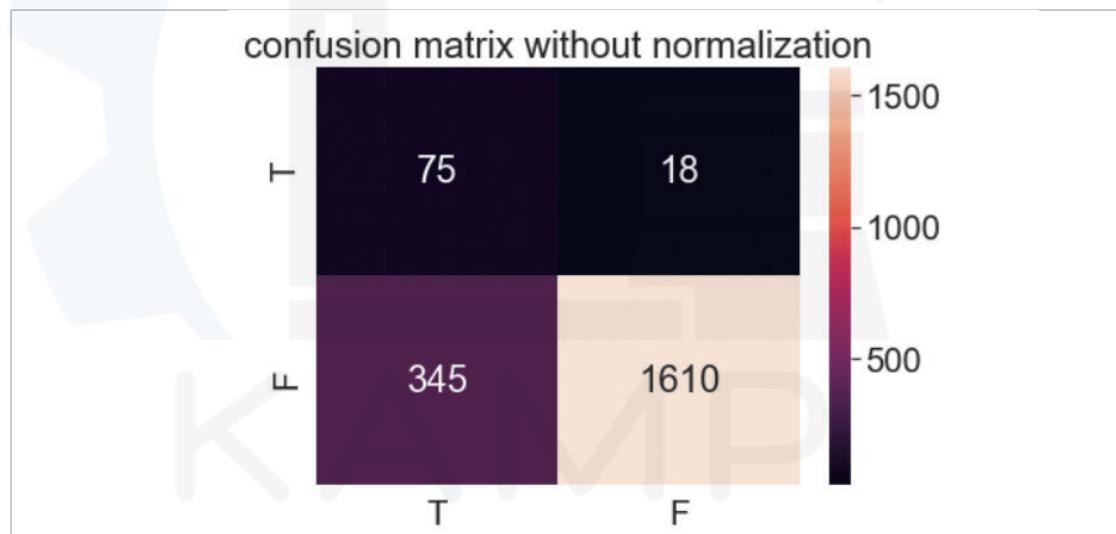
		Condition(실제)	
		Positive	Negative
Prediction (예측)	Positive	TP(True Positive)	FP(False Positive)
	Negative	FN(False Negative)	TN(True Negative)

[그림 18] 컨퓨전 매트릭스(Confusion Matrix)

- 실제(Condition)의 ‘Positive’는 실제 불량일 경우이며 ‘Negative’는 실제 불량이 아닌 경우이다. 예측(Prediction)의 ‘Positive’는 표준데이터를 이용하여 검출했을 때 불량인 경우와 ‘Negative’는 불량이 아닌 경우이다. 표의 각 항목에 대한 설명은 다음과 같다.

- \* TP : 참이라고 예측했으며 실제로도 참인 경우의 횟수
- \* FP : 참이라고 예측했지만 실제로 거짓인 경우의 횟수
- \* FN : 거짓이라고 예측했지만 실제로 참인 경우, 즉 거짓이라는 예측 결과가 틀린 횟수
- \* TN : 거짓이라고 예측한 결과가 실제로 거짓인 경우의 횟수

- 위의 4가지 지표를 통해 분류 성능 평가 척도인 정확도(Accuracy), 정밀도(Precision), 민감도(Sensitivity), 재현율(Recall), 특이도(Specificity)를 계산할 수 있다. 각 척도에 대한 설명은 다음과 같다.
- \* 정확도 : 참과 거짓이 모두 올바르게 분류된 확률  
 $= (TP+TN)/(TP+FP+FN+TN)$
- \* 정밀도 : 참으로 예측한 것 중 실제 참이 맞는 경우의 비율  
 $= TP/(TP+FP)$
- \* 민감도 : 실제 참인 경우 중 참으로 예측된 경우의 비율  
 $= TP/(TP+FN)$
- \* 재현율 : 민감도와 동일한 의미의 지표  
 $= TP/(TP+FN)$
- \* 특이도 : 실제 거짓인 경우 중 거짓으로 예측된 것의 비율  
 $= TN/(FP+TN)$



[그림 19] 히트맵을 이용한 분류 및 예측 결과 시각화

- 정확도는 분류 결과를 총체적인 관점에서 살펴보기에 적절하다. 하지만 참과 거짓을 구분하여 고려했을 때, 각각 잘 분류된 것인지 판단하기 어렵다. 따라서 정밀도, 민감도, 재현율, 특이도를 이용하여 분류 결과를 비교적 세밀하게 확인할 수 있다. 각 척도의 수치를 직접 제시할 수 있으며 [그림 19]와 같이 컴퓨터 매트릭스(Confusion matrix)를 히트맵으로 시각화하여 표현할 수도 있다.

(※ 히트맵 : 분류 및 예측 결과를 그래픽으로 출력하며 결과가 좋고 나쁜 정도를 색상으로 표현한다.)

- 여기서 주목할 점은, 최종적으로 각 성능 지표가 모두 높게 산출되는 것이 현실적으로 어렵다는 것이다. 민감도와 재현율은 동일한 의미이지만 이를 구분하여 생각한다면, 먼저 정밀도와 재현율은 서로 상충관계라고 할 수 있다. 만약 분류기가 실제로 참일 확률이 높은 것만 최종적으로 참이라 분류해낸다면, 정밀도와 재현율은 동시에 높아지기 어렵다. 정밀도는 참으로 예측된 것에 관한 비율이며 재현율은 실제 참인 것의 비율이기 때문이다.

- 이러한 문제를 해결하는 지표로 ‘F1-score’를 사용한다. F1-score는 정밀도와 재현율의 조화평균이다. 즉, 정밀도와 재현율을 동시에 고려한 지표로, F1-score가 높으면 두 지표의 상충관계가 일정 수준 이상 해결되어 분류가 잘 되었다고 판단할 수 있다. 이에 대한 계산식은 다음과 같다.

$$F1\text{-score} = 2 * \{(정밀도 * 재현율) / (정밀도 + 재현율)\}$$

- 따라서 이러한 개념을 바탕으로 각 성능 평가 척도의 수치를 직접적으로 제시하거나 시각화하여 표현하는 등의 방법을 이용하여 분류 성능을 평가할 수 있다.

#### • AI 분석 방법론(알고리즘) 구축 절차 설명

##### - 독립변수(시간) 설정

- 전체 데이터가 추출되는 일정한 패턴에 따라 한 번의 패턴이 시행되는 시간(예: 초 단위)을 파악한다. 이때의 시간에 맞추어 센서데이터 변수에 초 단위의 시간을 반복적으로 매칭(예: 1,024초 반복 매칭)시켜 K-최근접 이웃 회귀를 위한 독립변수(시간)를 생성한다. 즉, 독립변수는 시간이며 종속변수는 센서데이터로 하여 K-최근접 이웃 회귀에 적용한다.

##### - 시점별 예측값 추출

- 독립변수인 시간에 맞추어 K-최근접 이웃 회귀를 이용하여 센서데이터별 예측값을 추출한다. 시점별로 추출된 예측값을 표준데이터로 정의한다. 이때, 학습용 데이터와 별개로 분석 결과를 평가하기 위한 시험용 데이터는 분석 초반 과정에서 구분해 두도록 한다.

##### - 최소, 최대 임계치 추출

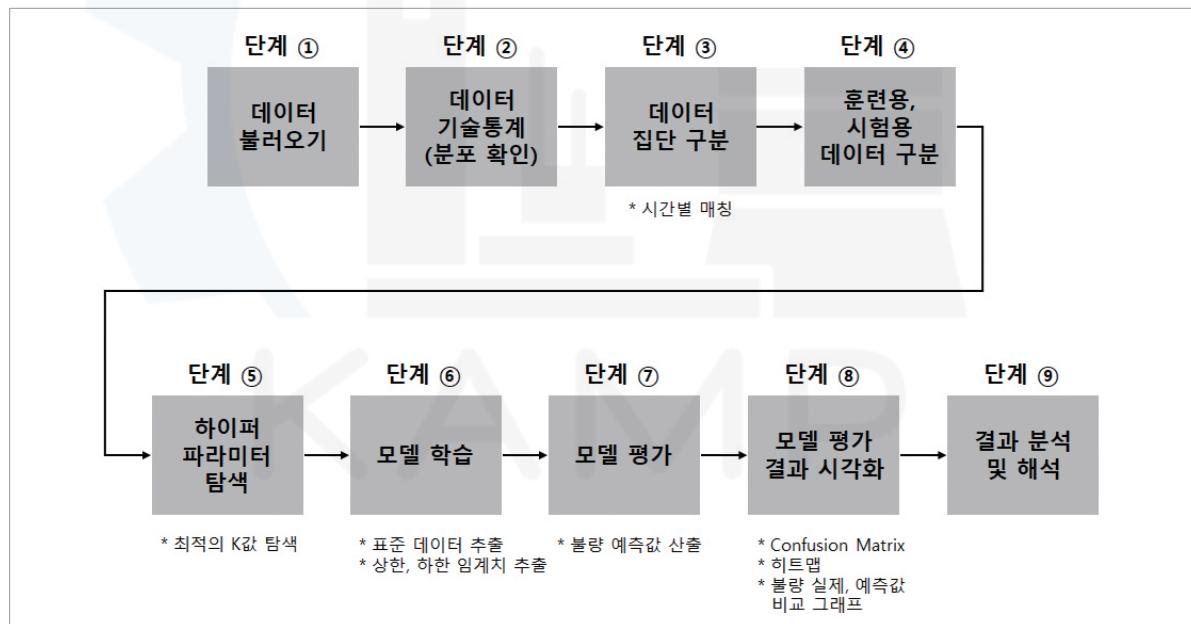
- 추출한 각 시점의 표준데이터에 대한 표준편차를 계산한다. 하나의 표준편차가 계산되면 모든 표준데이터에 ‘ $\pm 3$  표준편차’를 계산하여 표준데이터를 기반으로 한 최소 임계치와 최대 임계치를 산출한다.

## - 모델 평가

- 표준데이터와 최소, 최대 임계치가 불량 검출에 적합한지 검정한다. 이때, 학습에 사용된 데이터를 제외하고 시험용 데이터만 사용하도록 한다. 시험용 데이터에 표준데이터의 최소값과 최대값을 기준으로 확인한 불량 여부를 새롭게 매칭시킨다. 즉, 시험용 데이터 값이 최소값 미만이거나 최대값 초과일 경우를 불량으로 하여 불량 여부에 대한 예측값을 생성한다.
- 표준데이터로 검출된 불량이 실제 불량 여부와 얼마나 일치하는지 확인하여 표준데이터와 최소, 최대 임계치의 신뢰성을 확보한다. 불량과 불량이 아닌 경우가 균형을 이루며 일치하는지 확인하여 활용성 또한 확보한다. 이를 위해 컴퓨터 매트릭스를 시각화하고 정확도, 정밀도, 재현율, 민감도, 특이도와 F1-score를 산출한다.
- 이러한 검정 과정을 거쳐 최종적으로 가장 효과적인 표준데이터 추출을 위한 파라미터를 확인한다. 최종적으로 불량 여부에 대한 예측값과 실제값이 일치하거나 불일치하는 경우를 시각화하여 그래프에 표기한다.

## 2.3 분석 체험

### 1) 분석 단계별 Process



[그림 20] 분석 단계 Flow Chart

## 2) 분석 실습

### [단계 ①] 데이터 불러오기

#### ①-1. 필요 라이브러리 불러오기

```
| pip install pandas
```

[그림 21] pandas 라이브러리를 설치하기 위한 코드

- 라이브러리가 설치되지 않은 상태일 경우, '!pip install 라이브러리명'을 먼저 실행 시켜야 한다. 가장 먼저 pandas 라이브러리를 설치하기 위해 !pip install pandas 를 입력하고 실행한다. 진행 과정에서 사용하는 모든 라이브러리 중 설치하지 않은 것이 있다면 각 단계에서 위와 동일한 방법으로 모두 설치하면 된다.

(※ 라이브러리를 한번 설치하면 다시 설치하지 않아도 언제든지 불러올 수 있다. 코드를 종료하고 추후에 같은 PC로 다시 분석할 때도 마찬가지이다. 따라서 분석 과정에서는 각 라이브러리를 한번 설치하고 나서 다음 과정에서 다시 사용할 때 재설치하지 않는다.)

```
| import pandas as pd
```

[그림 22] pandas 라이브러리를 불러오기 위한 코드

- 설치가 완료되면 import pandas as pd를 입력하고 pandas 라이브러리를 불러온다. 이때 'as pd'라는 표현은 코드 내에서 pandas를 사용할 때 'pandas' 대신 'pd'라고 입력할 것이라고 선언하는 것이다.

(※ 라이브러리는 한번 불러오면 코드 파일을 종료하기 전까지 기능 수행이 유효하다. 따라서 앞쪽 단계에서 import 해두면 뒤쪽 단계에서 또다시 불러오지 않아도 된다. 단, 추후에 같은 PC로 분석을 할 때 라이브러리를 최초로 1번 불러와야 사용할 수 있다. 즉, 라이브러리 불러오기는 코드를 사용할 때 1번은 꼭 수행해야 한다.)

#### ②-2. 사용 데이터 불러오기(csv 파일)

```
| data = pd.read_csv(r'mixing_actuator.csv')
```

[그림 23] 데이터를 불러오기 위한 코드

- csv 파일을 불러올 때는 pandas의 read\_csv('파일 경로') 기능을 사용한다. pandas는 데이터를 기본 구조인 DataFrame으로 만든다.
- 'mixing\_actuator.csv'라는 데이터를 불러와서 data라는 객체에 할당한다. 변수에 데이터를 할당하면 코드 내에 객체명만 기입해도 데이터를 자유자재로 이용할 수 있다.
- 이때, notebook 파일(현재 사용하고 있는 코드 파일)과 데이터가 같은 위치에 존재할 경우 ('파일 경로') 내에 위와 같이 csv 파일명만 기재해도 오류가 발생하지 않는다. 단, 다른 위치에 존재할 경우, 예를 들어 ('C:/Users/Desktop/mixing\_



actuator.csv')와 같이 구체적인 데이터 파일의 경로를 기재해야 한다.

- 만약 [UnicodeDecodeError]가 발생할 경우, 변수명이나 데이터 내에 한글과 영어가 혼합되었기 때문일 확률이 높다. 따라서 pd.read\_csv(r'mixing\_actuator.csv', encoding='CP949') 혹은 pd.read\_csv(r'mixing\_actuator.csv', encoding='utf-8')으로 수정해야 한다. 오류를 방지하기 위해 가급적 변수명에는 한글이 포함되지 않도록 한다.

```
data
```

[그림 24] 데이터의 내용을 확인하기 위한 코드

	Unnamed: 0		Date	Sensor	Quality
0	0	2021-11-18T01:39:59.605000+00:00	0.064453	PASSED	
1	1	2021-11-18T01:39:59.605000+00:00	0.240234	PASSED	
2	2	2021-11-18T01:39:59.605000+00:00	-0.128906	PASSED	
3	3	2021-11-18T01:39:59.605000+00:00	-0.925781	PASSED	
4	4	2021-11-18T01:39:59.605000+00:00	-1.037109	PASSED	
...	...	...	...	...	...
351227	351227	2021-11-20T07:48:34.533000+00:00	0.262500	PASSED	
351228	351228	2021-11-20T07:48:34.533000+00:00	0.273438	PASSED	
351229	351229	2021-11-20T07:48:34.533000+00:00	1.796875	PASSED	
351230	351230	2021-11-20T07:48:34.533000+00:00	2.406250	FAILED	
351231	351231	2021-11-20T07:48:34.533000+00:00	0.742188	PASSED	

351232 rows × 4 columns

[그림 25] 데이터의 내용 확인 결과

- 불러온 데이터를 확인하기 위해 코드 실행줄에 데이터를 할당한 변수명인 data를 입력한다.
- 데이터 내에 Date, Sensor, Quality라는 3개의 변수가 포함되어 있는 것을 확인할 수 있다. Unnamed: 0 변수는 데이터를 불러올 때 자동으로 생성되는 인덱스 변수이다. 이 변수는 전처리 단계에서 삭제할 것이므로 나머지 변수들을 중심으로 확인한다. 현재 데이터는 351,232개의 행과 4개의 칼럼(변수)으로 이루어져 있다.

## [단계 ②] 데이터 기술통계

### ②-1. 필요 라이브러리 불러오기

```
!pip install matplotlib
```

[그림 26] matplotlib 라이브러리를 설치하기 위한 코드

- 먼저 !pip install matplotlib을 이용하여 필요한 라이브러리인 matplotlib을 설치한다.

```
import matplotlib.pyplot as plt
%matplotlib qt
```

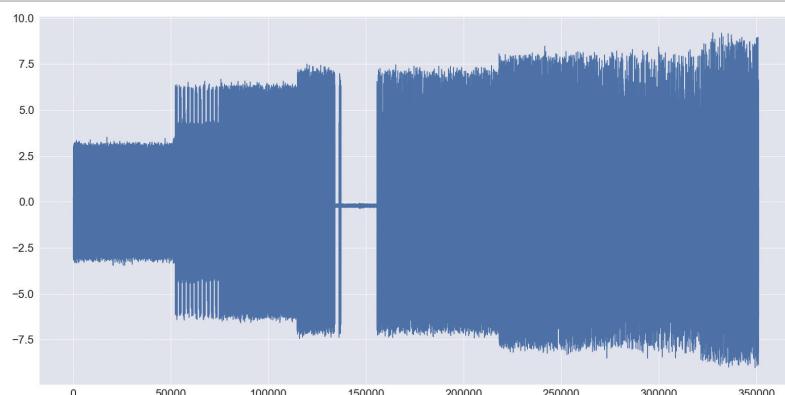
[그림 27] matplotlib 라이브러리를 불러오기 위한 코드

- 데이터의 분포를 확인할 그래프를 그릴 때 matplotlib.pyplot을 사용한다. matplotlib는 다양한 데이터를 많은 방법으로도 도식화할 수 있는 Python 라이브러리이다. 이를 이용하여 pandas에서 사용하는 자료 구조를 쉽게 시각화할 수 있다.
- 위의 코드를 직관적으로 해석하면, matplotlib의 pyplot을 이용해서 그래프를 그릴 것이므로 코드에 plt라고 기재한다면 matplotlib.pyplot의 기능을 사용할 수 있도록 하라는 의미이다.
- 또한, jupyter notebook에서 그래프를 제대로 확인하기 위해 %matplotlib qt라는 매직 커맨드를 작성해야 한다. 그래프를 출력할 때 새로운 창을 열고 그 안에 그래프를 표현하기 위해 사용한다. jupyter notebook 내에서 확인하면 화면 크기 조정에 제한이 있기 때문에 비교적 조정이 자유로운 새로운 창을 활용하도록 한다.

### ②-2. 데이터 분포 확인하기

```
plt.plot(data[ 'Sensor' ])
```

[그림 28] data 객체 내 Sensor 변수의 그래프를 그리기 위한 코드



[그림 29] Sensor 변수의 그래프를 그린 결과

- plt의 plot을 이용해서 data의 Sensor 변수의 분포를 확인한다. 코드를 시행했을 때 위와 같이 새로운 창이 뜨고 그 안에 그래프가 표현되는지 확인한다.
- 코드에서는 x축과 y축을 별도로 설정하지 않았다. 데이터는 시간에 따라 순차적으로 출력되었으므로 현재의 그래프에서 x축은 ‘시간’, ‘시점’, ‘시간의 흐름’으로 간주 한다. 따라서 y축은 센서 데이터(진동)의 값을 의미한다. 이 그래프를 통해 진동데이터가 어떠한 패턴을 띠는지 확인한다.

## ②-3. 데이터 통계량 확인하기

```
data['Sensor'].describe()
```

[그림 30] Sensor 변수의 요약통계량을 확인하기 위한 코드

```
count    351232.000000
mean      -0.108910
std       2.563446
min      -9.023438
25%     -1.992188
50%     -0.140625
75%      1.843750
max      9.203906
Name: Sensor, dtype: float64
```

[그림 31] Sensor 변수의 요약통계량 확인 결과

- 먼저 연속형(수치형)인 Sensor 변수의 기술통계를 확인한다. 이때 data의 Sensor 변수의 전체 개수, 평균, 표준편차, 최소값, 4분위수(25%, 50%, 75%), 최대값을 한 눈에 확인하기 위해 ‘변수명.describe()’를 사용한다. data['Sensor']는 ‘data라는 객체 내에 있는 Sensor 변수’를 의미한다. 만약 하나의 변수를 지정하지 않고 객체 내에 있는 모든 변수에 대한 기술통계를 확인하려면 data.describe()를 사용할 수 있다. 이 데이터에서는 연속형 변수 중 Sensor 데이터만 기술통계를 확인할 수 있으므로 변수 하나를 지정하였다.

```
data['Quality'].value_counts()
```

[그림 32] data 객체 내 Quality 변수의 범주별 개수 확인을 위한 코드

```
PASSED    191008
FAILED    160224
Name: Quality, dtype: int64
```

[그림 33] Quality 변수의 범주별 개수 확인 결과

- 다음으로 이산형(범주형)인 Quality 변수의 기술통계를 확인한다. 변수명.value\_counts()를 이용하면 해당 변수의 범주가 어떻게 분포되어 있는지에 대해 정량화

된 표현으로 확인할 수 있다. 위와 같이 현재 사용하는 데이터의 Quality 변수는 PASSED 값이 191,008개, FAILED 값이 160,224개로 이루어져 있다는 결과가 추출된다. PASSED 기준은 -2 이상 2 이하로 하였다. 그 외에 범위를 벗어나는 수치는 FAILED로 라벨링 하였다.

### [단계 ③] 데이터 집단 구분

#### ③-1. 불필요한 변수 사전 제거하기

```
data = data.drop(['Unnamed: 0'], axis=1) #axis=1은 열 기준 해당 변수를 삭제한다.
```

[그림 34] Unnamed: 0 칼럼을 삭제하기 위한 코드

- 데이터 파일을 한번 불러오면 인덱스 변수인 ‘Unnamed: 0’이 자동으로 생성된다. 분석 과정에서 변수를 포함시키지 않는 방법도 있지만 혼동이 올 수 있으므로 사전에 제거한다.
- 변수 제거를 위해 pandas에서 제공하는 drop 함수를 사용한다. drop 함수에 제거하고 싶은 ‘행 또는 열’의 이름과 위치를 입력한다. 즉, []내에 제거하고 싶은 행 또는 열의 이름을 적는다. axis가 0일 경우 행 기준, 1일 경우 열 기준으로 제거하라는 의미이다.
- 위의 코드는 변수를 삭제한 상태의 객체를 다시 동일한 객체인 data에 할당한다. 따라서 이후의 모든 과정에서 data 객체를 입력하면 Unnamed: 0이 삭제된 상태에서 정상적으로 사용할 수 있다.

#### ③-2. 추출 시간 기준 데이터 집단 구분하기

```
sec = [] #시간 리스트를 보관할 빈 리스트를 생성한다.

sec_num = 0
#for문(반복문)내에서 초기 설정하지 않아도 되도록 먼저 sec_num 변수를 지정한다.

for i in range(len(data)):
    sec_num += 1
    sec.append(sec_num)
    if sec_num == 1024: #1024초 간격으로 집단을 형성한다.
        sec_num = 0 #1024인 sec_num을 리스트에 넣고 나서 다시 0으로 리셋시킨다.
    else:
        sec_num = sec_num
```

[그림 35] 추출 시간을 기준으로 데이터 집단을 구분하기 위한 코드

- 위에서 알고리즘 구축 절차를 설명할 때, 센서데이터 변수에 초 단위의 시간을 반복적으로 매칭시켜 K-최근접 이웃 회귀를 위한 독립변수(시간)를 생성한다고 하였

다. 이를 위해 1,024초를 불량 여부를 체크하는 하나의 시간 집단이라고 가정하고, 1~1,024초가 데이터의 수대로 반복되도록 시간 집단을 생성한다.

- 위의 코드에서는 먼저 sec\_num=0을 통해 시간 리스트를 보관할 빈 리스트를 생성한다. 그다음 반복문을 시작하기 전에 sec\_num이라는 객체에 0을 할당함으로써 반복문 안에서 곧바로 sec\_num을 활용할 수 있도록 한다.
- for로 시작하는 코드는 for문(반복문)이라고 칭한다. Python은 데이터의 수를 0번 째부터 계산하기 때문에 range(len(data))를 설정하면 for문 아래의 코드가 ‘data의 길이-1’만큼 반복 수행된다. 코드가 구현되는 과정을 순차적으로 설명하자면 다음과 같다. sec\_num이 0인 상태에서 1을 더하고 그 숫자를 빈 리스트인 sec에 append 함수를 통해 보관한다. 단, 앞서 언급했듯이 한 집단은 1~1,024초까지이므로 if라는 조건문을 추가하여 sec\_num이 1,024가 되면 우선 그것을 sec 리스트에 보관한 후 sec\_num을 0으로 초기화한다. 초기화된 상태에서 다시 1을 더하여 sec에 보관한다. 이러한 과정을 data의 길이만큼, 즉 data 객체의 데이터 수만큼 1~1,024가 계속해서 반복되는 sec 리스트를 생성한다.

```
sec[1000:1050]
```

[그림 36] sec 리스트의 추출 결과 이상 여부를 확인하기 위한 코드

```
[1001,
1002,
1003,
1004,
1005,
1006,
1007,
1008,
1009,
1010,
1011,
1012,
1013,
1014,
1015,
1016,
1017,
1018,
1019,
1020,
1021,
1022,
1023,
1024,
1,
2,
3,
```

[그림 37] sec 리스트의 추출 결과를 확인한 결과

- sec 리스트가 잘 생성되었는지 확인한다. 숫자의 개수가 많으므로 1,024에서 1로 다시 초기화가 잘 되었는지 확인하기 위해 sec[1000:1050]으로 데이터를 잘라서 일부만 확인한다.

### ③-3. 데이터 집단과 예측 대상 데이터 매핑하기

```
data['sec'] = sec
```

[그림 38] sec 변수를 data 객체에 추가하기 위한 코드

- 분석 과정에서는 data 객체의 Sensor 변수를 종속변수로 하고 sec의 시간을 독립 변수로 할 것이므로 분석 과정의 편의성을 위해 sec 변수를 data 객체에 추가한다.
- data 객체에 sec 변수는 아직 존재하지 않지만 data['sec']=sec를 사용하여 변수를 추가할 수 있다. 위의 코드는 data 객체에 sec 변수를 새로 추가하며 이 값은 sec 리스트의 값들로 채운다는 의미이다.
- 단, 이렇게 곧바로 데이터프레임에 변수를 추가할 때는 객체의 길이와 추가하고자 하는 리스트의 길이가 동일해야 한다. 위에서 data 객체의 길이만큼 sec을 형성했기 때문에 이처럼 곧바로 변수 추가가 가능하다.

```
data[0:1025] #데이터를 슬라이스해서 확인한다.
```

[그림 39] data 객체의 일부 데이터를 확인하기 위한 코드

	Date	Sensor	Quality	sec
0	2020-11-18T20:09:20.039000+00:00	0.146094	PASSED	1
1	2020-11-18T20:09:20.039000+00:00	0.544531	PASSED	2
2	2020-11-18T20:09:20.039000+00:00	1.381250	PASSED	3
3	2020-11-18T20:09:20.039000+00:00	3.187500	PASSED	4
4	2020-11-18T20:09:20.039000+00:00	6.560937	FAILED	5
...	...	...	...	...
1020	2020-11-18T20:09:20.039000+00:00	-1.015625	PASSED	1021
1021	2020-11-18T20:09:20.039000+00:00	-0.406250	PASSED	1022
1022	2020-11-18T20:09:20.039000+00:00	-2.070312	PASSED	1023
1023	2020-11-18T20:09:20.039000+00:00	-2.734375	PASSED	1024
1024	2020-11-18T20:21:26.742000+00:00	-0.146094	PASSED	1

[그림 40] data 객체의 일부 데이터를 확인한 결과

- data 객체에 sec 변수가 잘 추가되었는지 확인한다. data[0:1025]로 data를 슬라이싱해서 확인했을 때, sec 변수가 추가된 모습을 확인할 수 있다.

#### [단계 ④] 훈련용/시험용 데이터 구분

##### ④-1. 필요 라이브러리 불러오기

```
!pip install numpy
!pip install scikit-learn
```

[그림 41] numpy와 scikit-learn 라이브러리를 설치하기 위한 코드

- 먼저 지금까지의 과정에서 설치하지 않았던 라이브러리인 numpy와 scikit-learn을 설치한다. !pip install numpy와 !pip install scikit-learn을 각각 입력한 후 실행하여 설치한다.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

%matplotlib qt
```

[그림 42] 필요 라이브러리를 불러오기 위한 코드

- 앞서 언급했듯이, 라이브러리를 불러올 때 import x as y를 사용하는 이유는 코드에서 더욱 편리하게 이용하기 위함이다. as 뒤에 코드 내에서 사용하고 싶은 라이브러리의 명칭을 직접 지정해준다.
- import numpy as np에서 numpy는 고성능의 수치계산을 위해 사용하는 라이브러리이다. 기본적으로 array(행렬) 단위로 데이터를 관리하며 이에 대해 연산을 수행한다. numpy를 사용할 때 numpy 대신 코드에서 np라고 기재하기 위해 사용한다.
- from sklearn.model\_selection import train\_test\_split은 머신러닝 모델을 학습하고 그 결과를 검증하기 위해 사용한다. 특히 원래의 데이터를 훈련, 검증, 테스트의 용도로 나누어 다루기 위해 scikit-learn에서 제공하는 model\_selection 모듈의 train\_test\_split 함수를 사용한다. 즉, train\_test\_split 함수를 이용하면 전체 데이터가 자동으로 훈련, 검증, 테스트로 분할된다.

\* scikit-learn : 지도학습, 비지도학습 모델 선택 및 평가 라이브러리이며 데이터 변환 및 데이터 불러오기와 계산 성능 향상에도 사용한다.)

#### ④-2. 독립/종속변수 설정하기

```
X = data['sec'] # 독립변수 : 매칭시킨 패턴 내 시간
y = data['Sensor'] # 종속변수 : 센서 데이터 1개의 변수
```

[그림 43] 독립변수 및 종속변수를 설정하기 위한 코드

- 훈련용, 시험용 데이터를 구분하기 위해 먼저 독립변수와 종속변수를 설정한다. 독립변수는 X, 종속변수는 y이다. 독립변수로는 앞 과정에서 매칭시켰던 패턴의 시간인 sec 변수를 사용한다. 종속변수로는 센서데이터인 Sensor 변수를 사용한다.

#### ④-3. 훈련용/시험용 데이터 구분하기

```
# 데이터셋을 훈련 세트와 테스트 세트로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=0, shuffle=False)

X_train = X_train.values
X_test = X_test.values

y_train = y_train.values
y_test = y_test.values

X_train = X_train.reshape(X_train.shape[0], 1)
X_test = X_test.reshape(X_test.shape[0], 1)
```

[그림 44] 훈련 데이터와 시험 데이터를 나누기 위한 코드

- 훈련용 X, y와 시험용 X, y를 각각 생성한다. X\_train(훈련용 X), X\_test(시험용 y), y\_train(훈련용 y), y\_test(시험용 y)가 각 명칭이다. test\_size는 전체 데이터 중 8:2의 비율로 훈련용, 시험용 데이터를 구분한다는 의미이다. random\_state를 이용하면 동일한 코드를 여러 번 수행해도 같은 결과를 도출할 수 있다. shuffle=False로 설정하면 전체 데이터를 분할할 때 랜덤으로 선택하여 분할하지 않고 순서대로 분할하게 된다. 예지보전에서는 시간의 흐름에 따라 추출되는 데이터에 대해 유연하게 불량 여부를 판단하는 것이 목적이므로 데이터를 랜덤으로 선택하지 않도록 한다.
- X\_train.value를 사용하면 기존에 X에 함께 할당되었던 인덱스를 제외하고 변수의 값만 X\_train에 할당된다. 그리고 X\_train.reshape을 통해 알고리즘을 구현할 때 X\_train을 적용할 수 있도록 형태를 변형시키고 X\_train에 다시 할당한다. 이러한 과정을 X\_test에도 동일하게 적용한다. y\_train과 y\_test에는 reshape 과정이 필요하지 않으므로 .values만 적용한다.

```
X_train
```

[그림 45] 훈련용 X 데이터의 내용을 확인하기 위한 코드

```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [ 10],
       [ 11],
       [ 12],
       [ 13],
       [ 14],
       [ 15],
       [ 16],
       [ 17],
       [ 18]],
```

[그림 46] 훈련용 X 데이터의 내용 확인 결과

```
y_train
```

```
101010 01010
0101010 10000
0000000 01000
```

[그림 47] 훈련용 Y 데이터의 내용을 확인하기 위한 코드

```
array([ 0.06445312,  0.24023438, -0.12890625, ..., -1.64765625,
       -1.26171875,  1.9        ])
```

[그림 48] 훈련용 Y 데이터의 내용 확인 결과

- 데이터 분할 결과의 예시로 X\_train과 y\_train을 확인하면 위와 같다.

## [단계 ⑤] 하이퍼 파라미터 탐색

### ⑤-1. 필요 라이브러리 불러오기

```
!pip install math
```

[그림 49] math 라이브러리를 설치하기 위한 코드

- 다음 과정에서 수치에 대해 제곱근을 하여 RMSE를 구할 것이므로 !pip install math를 이용해서 math 라이브러리를 설치한다.

(※ RMSE : 실제값과 예측값이 얼마나 차이가 나는지, 즉 실제값을 예측하는데 얼마나 많은 오류 또는 차이가 발생했는지 나타내는 값을 의미한다.)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn import neighbors
from math import sqrt
from sklearn.metrics import mean_squared_error
```

[그림 50] 필요 라이브러리를 불러오기 위한 코드

- from sklearn.neighbors import KNeighborsRegressor는 K-최근접 이웃 회귀 모델을 생성하기 위해 scikit-learn의 KNeighborsRegressor를 불러온다.
- from math import sqrt는 Python에서 사칙연산 외에 다양한 수학 계산을 할 수 있도록 하며 따로 설치하지 않아도 사용할 수 있는 Python 내장 라이브러리이다. 다음 과정에서 제곱근 연산을 수행할 것이므로 sqrt 함수를 지정하여 import 한다.
- from sklearn import neighbors를 통해 K-최근접 이웃 회귀 모델을 생성하는 과정 중 K(가까운 이웃 데이터의 수)를 정해줄 수 있다. 이러한 기능을 자동적으로 수행하기 위해 neighbors를 불러온다.
- from sklearn.metrics import mean\_squared\_error를 이용하여 모델의 예측 정확도를 확인하기 위해 표준편차와 동일한 RMSE(Root Mean Squared Error)를 계산할 수 있는 mean\_squared\_error 함수를 불러온다. 특정 수치에 대한 예측 정확도를 확인할 때, 단순히 정확도(Accuracy)만으로는 정확히 판단하기 어려우므로 RMSE 수치로 예측/분류 정확도를 판단한다. 따라서 RMSE 수치가 낮을수록 분석 모델의 성능이 좋다고 판단할 수 있다.

### ⑤-2. RMSE를 이용하여 최적의 K값 탐색하기

```
rmse_val = []

for K in range(17):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K,
                                           weights = 'distance')
    model.fit(X_train, y_train) #데이터를 모델에 적용한다.
    pred=model.predict(X_test) #모델의 학습 내용을 가지고 X_test로 예측한다.
    error = sqrt(mean_squared_error(y_test,pred)) #RMSE를 계산한다.
    rmse_val.append(error) #RMSE를 계속해서 보관해둔다.
print('RMSE value for k= ', K , 'is:', error)
```

[그림 51] K값별 RMSE를 통해 최적의 K값을 탐색하기 위한 코드

```

RMSE value for k= 1 is: 3.526465274641486
RMSE value for k= 2 is: 3.098809867069936
RMSE value for k= 3 is: 2.8832316276264844
RMSE value for k= 4 is: 2.793099876492619
RMSE value for k= 5 is: 2.727928597718662
RMSE value for k= 6 is: 2.689560100101723
RMSE value for k= 7 is: 2.6707263806291013
RMSE value for k= 8 is: 2.646891087495563
RMSE value for k= 9 is: 2.6306180874253773
RMSE value for k= 10 is: 2.6138332052434037
RMSE value for k= 11 is: 2.6037293590713535
RMSE value for k= 12 is: 2.592108954565333
RMSE value for k= 13 is: 2.5850699965951027
RMSE value for k= 14 is: 2.5764959455111422
RMSE value for k= 15 is: 2.5719649975045105
RMSE value for k= 16 is: 2.5660851411415013
RMSE value for k= 17 is: 2.562028123496635

```

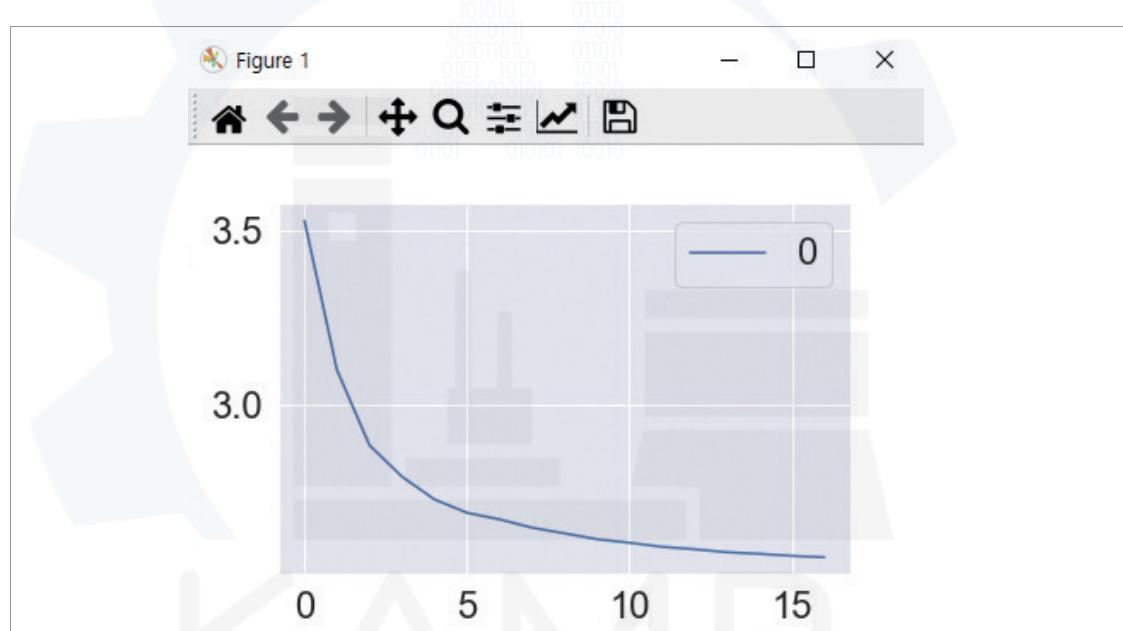
[그림 52] RMSE를 통해 최적의 K값을 탐색한 결과

- RMSE는 점차 감소하다가 K가 17일 때 가장 낮은 값을 보이는 것을 확인할 수 있다. 따라서 최적의 K를 17로 설정할 수 있다.  
\* K-최근접 이웃 알고리즘은 다수결의 개념이 내포되어 있으므로 K를 짹수로 지정하는 것은 지양한다.)
- 위 코드에 대한 직관적이고 순차적인 해석은 다음과 같다.
- 최적의 K를 찾기 위해 K마다 얼마만큼의 RMSE가 계산되는지 우선 비교해야 한다. 따라서 RMSE 값들을 하나의 리스트에 보관하고 비교하기 위해 rmse\_val이라는 빈 리스트를 생성한다.
- 반복문에서는 반복할 횟수의 범위를 지정하면 0부터 시작하여 ‘직접 기재한 숫자-1’까지의 범위에서 명령이 반복 수행된다고 하였다. 위의 코드에서는 K에 1을 더 해가며 이웃 수로 지정해보고 이러한 과정을 1~17까지 반복 수행할 수 있도록 한다.  
\* Python은 0부터 시작하여 범위를 지정하므로 0번째가 1번째를 의미한다.)
- 해당 과정에서 지정한 K를 K-최근접 이웃 회귀에 적용하기 위해 neighbors.KNeighborsRegressor( )를 이용한다. 이 명령에서 사용할 수 있는 옵션 중 n\_neighbors는 K로 지정함으로써 코드의 반복 수행에 따라 이웃 수를 자동으로 할당 한다. 또한, 가중평균을 적용하기 위해 weights='distance'로 지정한다. 이러한 기능을 가진 하나의 모델을 model 객체에 할당한다.  
\* 가중평균은 거리가 가까울수록 데이터가 더 유사할 것이라는 가중치를 부여한다.)
- 앞에서 분할한 훈련용 데이터 X와 y를 모델에 적용시키기 위해 model.fit( )을 사용 한다. 즉, K-최근접 이웃 회귀의 기능을 model에 fit 시킴으로써 model 객체는 학습한(인지한, 습득한) 내용을 기억하게 된다.
- 모델이 학습한 내용을 바탕으로, 시험데이터 X로 시험데이터 y를 예측하라고 명령 한 후의 예측 결과를 pred 변수에 저장하기 위해 model.predict( )를 사용한다.

- mean\_squared\_error() 함수를 사용하여 시험데이터 y와 예측된 값 y 간의 RMSE를 계산하고 이를 error 객체에 저장한다. 이때, error 값은 1을 더한 K를 새롭게 적용하는 단계에서 또다시 새로운 error 값이 할당되므로 이전의 error 값을 기억하지 않고 계속해서 새로운 값으로 바뀌게 된다.
- error 변수가 초기화되기 전에 계속해서 rmse\_val 리스트에 보관해두기 위해, rmse\_val.append() 함수를 사용한다. 이때, 비어있는 rmse\_val 리스트를 만드는 명령은 for문이 시행되기 이전에 수행되므로 K가 반복적으로 업데이트되는 과정 중에 초기화되지 않는다. 따라서 새로운 error값이 계속해서 추가될 수 있다.
- 어떤 K일 때 얼마큼의 RMSE가 계산되는지 출력하기 위해 print() 함수를 사용한다.

```
curve = pd.DataFrame(rmse_val)
curve.plot()
```

[그림 53] K값별 RMSE를 데이터프레임 형태로 변환 후 그래프를 그리기 위한 코드



[그림 54] K값별 RMSE 계산 결과를 그래프로 그린 결과

- RMSE 리스트인 rmse\_val의 변화 양상을 확인하기 위해 이를 pandas의 DataFrame() 함수를 이용하여 데이터프레임 형태로 만든다. 데이터프레임은 curve 변수에 할당하고 plot() 함수를 이용하여 RMSE의 변화 양상을 그래프로 확인한다. K가 증가할수록 RMSE가 감소하면서 일정하게 낮은 값을 유지하는 것을 확인할 수 있다.

## [단계 ⑥] 모델 학습

### ⑥-1. 표준데이터 추출하기

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[그림 55] sigmoid 함수를 정의하기 위한 코드

- K-최근접 이웃 회귀 결과에서 모델의 설명력을 시그모이드 함수를 이용하여 계산하기 위해 sigmoid()라는 함수를 자체적으로 설정한다. 시그모이드 함수는 0과 1 사이의 실수로 입력값을 변환해주는 함수이다. 이처럼 def를 이용하여 직접 함수를 만들면 sigmoid(x)를 코드에 입력했을 때, 함수식에 x가 적용된 결과가 출력된다. x는 적용하고자 하는 변수의 이름으로 기재하면 된다. 자체적으로 함수를 생성하면 메인 코드의 세부 사항이 변동될 때 해당 메인 코드에서 내용을 매번 변경하지 않아도 되며 함수의 내용만 변경하면 된다는 장점이 있다.

```
#적합한 K를 적용하여 reg.predict(line)를 수행한다.
line = np.linspace(1, 1024, 1024).reshape(-1, 1)

reg = KNeighborsRegressor(n_neighbors=17, weights = 'distance')
reg.fit(X_train, y_train)
standard = reg.predict(line)
print(standard)
print(len(standard)) #표준 데이터의 길이를 확인한다.
```

[그림 56] K-최근접 이웃 회귀를 이용하여 표준데이터를 추출하기 위한 코드

```
[-0.01017923  0.15969669  0.15372243 ...  0.30491728 -0.45473346
 0.08570772]
1024
```

[그림 57] 추출한 표준데이터와 개수(길이)를 확인한 결과

- 데이터셋에 대해 가능한 많은 특성값을 만들어 예측할 수 있다. 이를 위해 x축을 따라 많은 포인트(점)를 생성하여 테스트 데이터셋(line)을 만든다. 즉, 예측이 잘 이루어졌는지 확인하고 표준데이터를 추출하기 위해 별도의 데이터셋(line)을 만든다. 1부터 a 사이에 b개의 데이터 포인트를 만들기 위해 np.linspace(1, 1024, 1024)를 수행한다. reshape를 이용하여 predict에 적합한 형태로 변환시킨다.
- 위에서 최적의 K로 추출된 17을 n\_neighbors에 할당하여 K-최근접 이웃 회귀의 K를 17로 지정한다. 가중평균을 적용하기 위해 KNeighborsRegressor()의 옵션인 weights에 'distance'를 사용한다. reg 객체에 K-최근접 이웃 회귀 모델의 옵션을 모두 지정하고 그 내용을 바탕으로 X\_train과 y\_train을 훈련시키기 위해 reg.fit(X\_train, y\_train)을 사용한다. 이 과정을 거치면 reg 객체에는 X\_train과 y\_train이 학습한 내용이 저장된다.

- reg 객체는 학습 내용을 지니고 있는 상태이므로 앞 단계에서 생성한 데이터 포인트 (line)를 바탕으로 새로운 y값을 예측하기 위해 reg.predict(line) 명령어를 입력한다. 따라서 이 단계에서 추출된 예측값은 standard 객체에 저장되므로 1024개의 표준데이터는 모두 standard 객체 안에 보관되어 있다고 간주한다. print(standard)와 print(len(standard))를 통해 간략한 표준데이터의 값과 총 길이(개수)를 확인할 수 있다.

#### ⑥-2. 상한/하한 임계치 추출하기

```
#표준 데이터의 표준편차를 추출한다.
sigma = np.std(standard)
print(sigma) #표준편차를 확인한다.
```

[그림 58] 표준데이터의 표준편차를 추출하기 위한 코드

0.6044158707707497

01010  
01010101  
10010

[그림 59] 표준데이터의 표준편차 추출 결과

- 표준데이터를 추출한 후 상한, 하한 임계치를 추출하기 위해 표준편차를 계산해야 한다. 상한 임계치는 ‘표준데이터+3\*표준편차’이며 하한 임계치는 ‘표준데이터-3\*표준편차’로 설정할 것이다.
- np.std() 함수를 이용하여 standard 객체에 저장된 모든 값을 바탕으로 표준편차를 계산하고 이를 통해 추출된 하나의 표준편차 값을 sigma 객체에 할당한다. 따라서 sigma 객체에는 위의 코드 수행 결과와 같이 약 0.604의 값 하나가 저장된다.

```
#하한 임계치와 상한 임계치를 생성한다.
min_standard = standard - 3*sigma #하한 임계치
max_standard = standard + 3*sigma #상한 임계치
```

[그림 60] 표준편차를 기준으로 하한 및 상한 임계치를 생성하기 위한 코드

- 하나의 집단을 데이터 추출 기준인 1,024초로 설정하였으므로 총 1,024개의 데이터(표준데이터의 개수와 동일)에 대한 상한, 하한 임계치를 계산한다. 예를 들어, standard에는 현재 1,024개의 표준데이터가 저장되어 있는데 standard-3\*sigma를 계산하면 1,024개의 데이터 각각의 값에 3\*sigma를 계산한 값이 빼어지게 된다. 모든 데이터에 동일한 값을 빼려 한다면 위와 같이 간단히 한 줄로 표현할 수 있다. 이와 동일한 방법으로 상한 임계치도 추출한다. 위의 코드를 수행하면 min\_standard 객체에는 하한 임계치 1,024개가 저장되며 max\_standard 객체에는 상한 임계치 1,024개가 저장된다.

```
#y_test 길이만큼 최소 임계치를 반복시킨다.
min_standard_list = []
j = 0

for i in range(len(y_test)):
    min_standard_list.append(min_standard[j])
    j += 1
    if j == len(min_standard):
        j = 0 #초기화한다.
```

[그림 61] 실제 y값의 길이만큼 하한 임계치 집단을 반복하여 매칭하기 위한 코드

- 상한, 하한 임계치를 추출하고 나면 지금까지의 과정에서 사용하지 않았던 실제 y값과 임계치를 통한 예측 y값의 일치 여부를 비교해야 한다.
- 이를 위해, 먼저 위의 코드와 같이 하한 임계치인 min\_standard가 실제 y값의 수만큼 반복되게 하여 두 변수가 동일한 길이로 이루어지도록 계산해야 한다. 1,024초의 시간 집단을 데이터의 총 개수만큼 반복되게 매칭했던 원리와 같다.
- min\_standard\_list라는 빈 리스트를 생성하고 min\_standard 값을 순서대로 반복하면서 이 리스트에 저장한다. 실제 y값이 들어있는 y\_test 객체의 길이가 min\_standard의 길이보다 길기 때문에 반복문이 수행되는 횟수는 y\_test의 길이를 기준으로 하여 len(y\_test)로 입력한다. i는 0부터 70247(y\_test의 길이)까지 순차적으로 지정되며 이는 그 수만큼 for문 안의 명령을 반복하라는 의미이다. for문이 시행되기 전에 0으로 지정한 j는 다음과 같은 방법으로 사용한다. min\_standard[j]는 예를 들어 j가 0일 경우 min\_standard의 0번째 수를 지정한다는 의미이다. 위의 코드에서는 min\_standard\_list에 min\_standard를 계속해서 지정하여 저장하며 j는 하나의 min\_standard 수가 list에 저장될 때마다 '+= 1'을 통해 1씩 늘어난다. if문에서는 만약 j가 1씩 늘어나다가 min\_standard의 길이와 동일해진다면 1씩 더하는 것을 멈추고 0으로 초기화 할 것을 명령한다.

```
#y_test 길이만큼 최대 임계치를 반복시킨다.
max_standard_list = []
j = 0

for i in range(len(y_test)):
    max_standard_list.append(max_standard[j])
    j += 1
    if j == len(max_standard):
        j = 0 #초기화한다.
```

[그림 62] 실제 y값의 길이만큼 상한 임계치 집단을 반복하여 매칭하기 위한 코드

```
#y_test 길이만큼 표준 데이터를 반복시킨다.
standard_list = []
j = 0

for i in range(len(y_test)):
    standard_list.append(standard[j])
    j += 1
    if j == len(standard):
        j = 0 #초기화한다.
```

[그림 63] 실제 y값의 길이만큼 표준데이터 집단을 반복하여 매칭하기 위한 코드

- 상한 임계치와 표준데이터도 y\_test와 같은 데이터프레임에 위치시키기 위해 위와 동일한 과정을 거친다.

```
df = pd.DataFrame({'real_y':y_test, 'standard':standard_list,
                   'min_standard':min_standard_list,
                   'max_standard':max_standard_list})
```

[그림 64] 실제 y값, 표준데이터, 하한 임계치와 상한 임계치를 묶어 하나의 데이터프레임으로 만들기 위한 코드

```
df
```

[그림 65] 데이터프레임의 내용을 확인하기 위한 코드

	real_y	standard	min_standard	max_standard
0	0.638281	-0.010179	-1.823427	1.803068
1	-0.519531	0.159697	-1.653551	1.972944
2	0.519531	0.153722	-1.659525	1.966970
3	0.207813	-0.054090	-1.867338	1.759158
4	-2.404688	-0.802045	-2.615293	1.011203
...	...	...	...	...
70242	0.262500	-0.182008	-1.995256	1.631239
70243	0.273438	-0.549104	-2.362351	1.264144
70244	1.796875	-0.878929	-2.692177	0.934318
70245	2.406250	-1.459881	-3.273128	0.353367
70246	0.742188	0.079802	-1.733445	1.893050

70247 rows × 4 columns

[그림 66] 데이터의 내용 확인 결과

- 위의 과정을 마친 후 실제 y값, 표준데이터, 하한 임계치, 상한 임계치를 하나의 데이터프레임에 모아 df라는 객체에 저장한다. df를 입력해보면 위와 같이 합쳐진 데이터프레임을 확인할 수 있다.

## [단계 ⑦] 모델 평가

### ⑦-1. 불량 예측값 산출하기

```
#하한, 상한 임계치를 y_test와 비교하여 예측 품질 컬럼을 추가한다.
pred_fault = []
for i in range(len(df)):
    if df['real_y'][i] >= df['max_standard'][i] or df['real_y'][i] <= df['min_standard'][i]:
        pred_fault.append(1)
    else:
        pred_fault.append(0)
```

[그림 67] 실제 y값과 하한, 상한 임계치를 비교하여 예측 불량 여부를 추출하고 새로운 리스트에 저장하기 위한 코드

- df 객체 안에 있는 데이터프레임에 pred\_fault 변수를 추가하기 위해 먼저 pred\_fault 리스트를 생성한다. 이 리스트에는 실제 y값인 df 객체의 real\_y 변수 내 각각의 값들이 max\_standard 변수 내의 값보다 크거나 min\_standard 변수 내의 값보다 작을 경우 1의 값이 입력된다. 즉, 실제 y값이 상한 임계치보다 크거나 하한 임계치보다 작을 경우 1, 그 반대일 경우 0으로 이루어져 표준데이터를 이용했을 때 불량 여부 예측 결과에 대한 변수이다. 예측된 분류 결과에서 1이면 불량, 0이면 정상임을 의미한다.
- df 객체의 real\_y 변수와 max\_standard, min\_standard 변수를 비교한 결과를 산출하기 위해 for문을 사용하고 .append() 함수를 이용하여 pred\_fault 리스트에 1 또는 0을 계속해서 저장한다.

```
real_fault = []
for i in range(len(df)):
    if data['Quality'][280985+i] == 'FAILED':
        real_fault.append(1)
    else:
        real_fault.append(0)
```

[그림 68] 원데이터에서 실제 y값에 대한 불량 여부를 추출하여 새로운 리스트에 저장하기 위한 코드

- 원래의 데이터(data 객체)에서는 Sensor 변수와 Quality 변수가 함께 있었다. 현재 df 객체에는 실제 y값이 불량 혹은 정상을 의미하는 것인지에 대한 Quality 변수가 누락되어 있다. 따라서 위의 코드를 통해 df 객체의 길이를 기준으로 df 객체의 Quality 값이 FAILED일 경우 1, PASSED일 경우 0으로 매칭하여 real\_fault 리스트를 생성한다. 위의 코드에서 shuffle=False로 설정했었다. 따라서 df 객체에 존재하는 70247개의 y\_test 값은 전체 데이터인 data 객체를 원래 순서대로 나열했을 때 맨 뒤에서부터 70247개의 데이터와 동일하다.
- 따라서 data['Quality'][i]를 if 조건문에 적용하는 것이 아니라 data['Quality'][280985+i]로 설정하여 280,985번째 데이터부터 순서대로 비교한다. 즉, df 객체의 y\_test값과 data 객체의 y\_test값이 동일한 상태에서의 Quality를 확인해야 하므로 위와 같이 설정해야 한다.

## ⑦-2. 분석 결과를 반영한 최종 데이터프레임 형성하기

```
df['pred_fault'] = pred_fault
df['real_fault'] = real_fault
```

[그림 69] 예측 불량 여부와 실제 불량 여부 리스트를 데이터프레임에 추가하기 위한 코드

```
df
```

[그림 70] 데이터프레임의 변수 추가 내용을 확인하기 위한 코드

	real_y	standard	min_standard	max_standard	pred_fault	real_fault
0	0.638281	-0.010179	-1.823427	1.803068	0	0
1	-0.519531	0.159697	-1.653551	1.972944	0	0
2	0.519531	0.153722	-1.659525	1.966970	0	0
3	0.207813	-0.054090	-1.867338	1.759158	0	0
4	-2.404688	-0.802045	-2.615293	1.011203	0	1
...	...	...	...	...	...	...
70242	0.262500	-0.182008	-1.995256	1.631239	0	0
70243	0.273438	-0.549104	-2.362351	1.264144	0	0
70244	1.796875	-0.878929	-2.692177	0.934318	1	0
70245	2.406250	-1.459881	-3.273128	0.353367	1	1
70246	0.742188	0.079802	-1.733445	1.893050	0	0

70247 rows × 6 columns

[그림 71] 데이터프레임의 변수 추가 내용을 확인한 결과

- df 객체에 pred\_fault 변수와 real\_fault 변수를 새로 생성한다.
- 'df'를 코드 실행줄에 입력해보면 위와 같은 데이터프레임을 확인할 수 있다. 이전의 df 객체에 pred\_fault와 real\_fault 변수가 정상적으로 추가되었다.

## [단계 ⑧] 모델 평가 결과 시각화

### ⑧-1. 필요 라이브러리 불러오기

```
!pip install seaborn
```

[그림 72] seaborn 라이브러리를 설치하기 위한 코드

- 히트맵 시각화를 위한 seaborn 라이브러리를 미리 설치하기 위해 !pip install seaborn을 입력하고 실행한다.

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
sns.set(font_scale=2)
```

[그림 73] 필요 라이브러리를 불러오기 위한 코드

- 위에서 언급했듯이 분류 성능을 평가하는 척도에는 정확도(accuracy), 정밀도(precision), 재현율(recall)과 f1-score가 있다. scikit-learn에서 제공하는 accuracy\_score, recall\_score, f1\_score 라이브러리를 활용하면 코드 실행줄에서 해당 척도가 정량적으로 얼마나 산출되었는지 확인할 수 있다. 마찬가지로 confusion\_matrix와 classification\_report 라이브러리를 이용하면 위의 척도를 포함한 분류 결과를 한눈에 확인할 수 있다.
- 분류 결과를 히트맵으로 표현하기 위해 seaborn의 sns를 불러오며 font\_scale을 설정함에 따라 히트맵 내에 표현되는 글자와 숫자의 크기를 지정한다.

### ⑧-2. 실제 센서데이터 값과 표준데이터 값의 비교 그래프 그리기

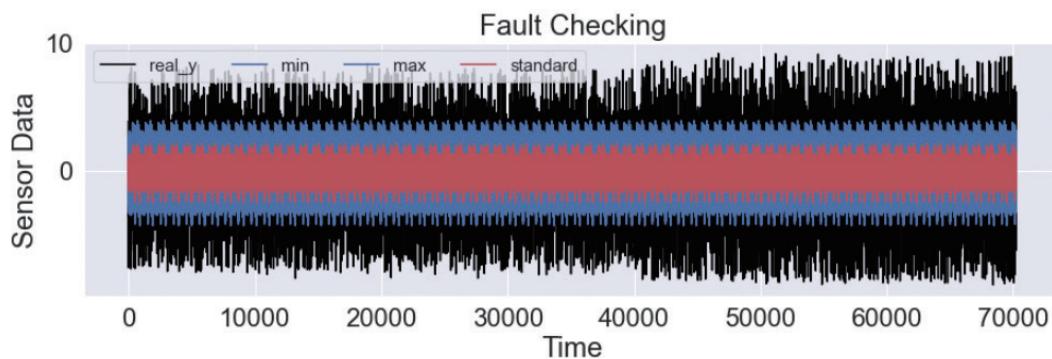
```
plt.figure(figsize=(15,4))

plt.plot(df['real_y'], 'black', label='real_y')
plt.plot(df['min_standard'], 'b', label='min')
plt.plot(df['max_standard'], 'b', label='max')
plt.plot(df['standard'], 'r', label='standard')

plt.xlabel('Time')
plt.ylabel('Sensor Data')
plt.title('Fault Checking')

plt.legend(loc='best', ncol=4, fontsize=15)
plt.show()
```

[그림 74] 실제 센서데이터, 표준데이터 및 임계치를 하나의 그래프에 그리기 위한 코드



[그림 75] 실제 센서데이터, 표준데이터 및 임계치를 하나의 그래프에 그린 결과

- 실제 센서데이터(y) 값과 표준데이터, 하한 임계치, 상한 임계치를 그래프에 함께 표현한다. 실제 y값(real\_y)은 검정색 선, 파란색 선은 아래쪽과 위쪽으로 각각 하한(min), 상한(max) 임계치이며 빨간색 선은 표준데이터(standard)를 의미한다. x축은 앞에서 설정한 1,024초가 반복되는 형태이며 y축은 센서데이터의 값이 표시되었다. 시각적으로 어떤 시점의 값이 표준데이터, 상한, 하한 임계치의 범위를 벗어나고 있는지 확인할 수 있다.

### ⑧-3. 분류 성능 평가 척도 산출하기

```
print('정밀도는', round(precision_score(df['real_fault'], df['pred_fault']), 2)*100, '% 입니다')
print('정확도는', round(accuracy_score(df['real_fault'], df['pred_fault']), 2)*100, '% 입니다')
print('재현율은', round(recall_score(df['real_fault'], df['pred_fault']), 2)*100, '% 입니다')
print('f1-score는', round(f1_score(df['real_fault'], df['pred_fault']), 2)*100, '% 입니다')
```

[그림 76] 정밀도, 정확도, 재현율, f1-score를 산출하기 위한 코드

```
정밀도는 80.0 % 입니다
정확도는 87.0 % 입니다
재현율은 92.0 % 입니다
f1-score는 86.0 % 입니다
```

[그림 77] 정밀도, 정확도, 재현율, f1-score 산출 결과

- 위의 코드에서는 분류 성능 평가 척도인 정밀도, 정확도, 재현율과 f1-score를 확인하기 위해 precision\_score(실제값, 예측값), accuracy\_score(실제값, 예측값), recall\_score(실제값, 예측값), f1\_score(실제값, 예측값)를 사용하였다. round( ,2) 를 입력한 이유는 출력되는 실수를 소수점 둘째 자리까지 표현하기 위함이다. 그리고 %로 표현하기 위해 추출된 척도 값에 100을 곱하였다. 코드를 실행했을 때 출력되는 결과를 통해 정밀도는 80%, 정확도는 87%, 재현율은 92%이며 f1-score는 86%인 것을 알 수 있다.

```
print(confusion_matrix(df['real_fault'], df['pred_fault'], labels=[1,0]))
print(classification_report(df['real_fault'], df['pred_fault'], target_names=['정상', '불량']))
```

[그림 78] 컴퓨전 매트릭스와 분류 결과를 산출하기 위한 코드

	precision	recall	f1-score	support
정상	0.94	0.84	0.89	41278
불량	0.80	0.92	0.86	28969
accuracy			0.87	70247
macro avg	0.87	0.88	0.87	70247
weighted avg	0.88	0.87	0.87	70247

[그림 79] 컴퓨전 매트릭스와 분류 결과를 산출하기 위한 코드

- confusion\_matrix()를 이용하면 위와 같이 분류 결과를 나타내는 행렬을 확인 할 수 있다. labels=[1, 0]으로 지정했기 때문에 행렬의 열 방향은 순서대로 예측값이 1인 경우와 0인 경우이다. 행 방향은 순서대로 실제값이 1인 경우와 0인 경우이다. 즉, 전체 데이터 중 실제 1(불량)인 경우는 총  $26,707 + 2,262 = 28,969$ 개로 계산할 수 있으며 모델을 이용한 예측 결과가 1(불량)인 경우는 총  $26,707 + 6,658 = 33,365$ 개로 계산할 수 있다. 반대로 전체 데이터 중 실제 0(정상)인 경우는  $6,658 + 34,620 = 41,278$ 개이다. 또한, 모델을 이용한 예측 결과가 0(정상)인 경우는  $2,262 + 34,620 = 36,882$ 개임을 알 수 있다.
- 각 항목별로 결과를 해석하면, 실제 불량인 경우를 분석 모델이 불량으로 정확히 예측해낸 경우는 26,707개이며 실제 불량이지만 분석 모델이 정상으로 예측한 경우는 2,262개이다. 마지막으로 실제 정상인 경우를 모델이 정상으로 올바르게 예측해낸 경우는 34,620개이며 실제 정상인 경우를 불량으로 예측한 경우는 6,658개이다. confusion\_matrix()를 이용하면 이처럼 불량 혹은 정상을 모델이 어떻게 예측해내었는지를 절대적인 수치로 확인할 수 있다. 하지만 분석 목적에 따라, 절대적인 수치가 아닌 전체 대비 비율을 계산해야 할 경우 classification\_report()를 추가로 수행해야 한다.
- classification\_report()를 이용하면 불량과 정상 라벨에 대하여 각 분류 성능 평가 척도가 어느 정도 산출되었는지 한눈에 확인할 수 있다. 코드를 수행한 결과를 해석하면 다음과 같다. 표준데이터와 상한, 하한 임계치는 실제 데이터의 불량인 경우 중 92%를 불량이라고 예측(recall 칼럼에서 불량 부분에 해당)한다. 또한, 불량으로 예측한 것 중 80%가 실제로 불량(precision 칼럼에서 불량 부분에 해당)임을 알 수 있다. 전체적으로 실제 불량인 데이터에 대해 재현율(recall)과 정밀도(precision)가 얼마나 균형을 이루었는지 확인해보면, 불량에 대한 f1-score가 86%로 산출되어 일정 수준 이상 균형을 이루고 있음을 알 수 있다.
- 유사한 방식으로 실제 정상인 데이터에 대한 결과를 해석하면, 분석 모델은 84% 수준의 성능으로 실제 정상인 경우를 정확히 정상으로 분류(recall 칼럼에서 정상 부분에 해당)하였으며 정상으로 예측한 것 중 94%가 실제로 정상(precision 칼럼에서 정상 부분에 해당)이었다. 실제 정상인 경우에 대한 f1-score는 89%로 산출되어 이때의 재현율과 정밀도가 일정 수준 이상 균형을 이루고 있음을 알 수 있다.
- 즉, K-최근접 이웃 회귀를 통해 실제 데이터가 불량인 경우와 정상인 경우 중 어느 한쪽에 치우치지 않고 모두 적절히 분류해내었음을 확인할 수 있다.

### ⑧-4. 컴퓨전 매트릭스(Confusion Matrix) 및 히트맵(Heat Map) 시각화하기

```
arr = [[26707, 2262],
       [6658, 34620]]
df_cm = pd.DataFrame(arr, index=[i for i in 'TF'], columns = [i for i in 'TF'])
```

[그림 80] 컴퓨전 매트릭스를 데이터프레임 형태로 변환하기 위한 코드

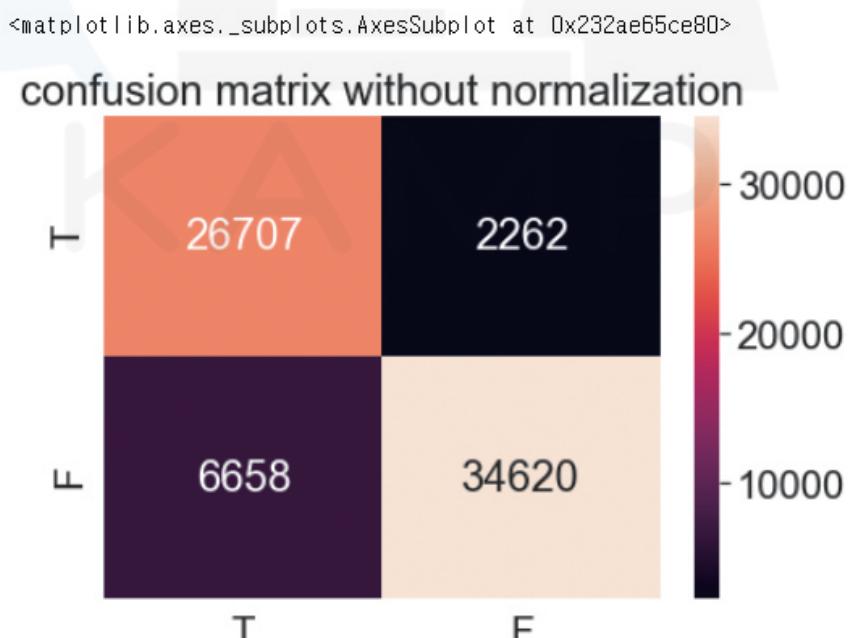
	T	F
T	26707	2262
F	6658	34620

[그림 81] 컴퓨전 매트릭스를 데이터프레임 형태로 변환한 결과

- 추출한 컴퓨전 매트릭스를 바탕으로 히트맵을 만들기 위해 매트릭스 추출 결과를 직접 데이터프레임 형식으로 변환한다. 행렬(배열) 형태였던 컴퓨전 매트릭스 추출 결과를 데이터프레임 형태로 변환하여 df\_cm 객체에 저장한다. index = [i for i in 'TF']로 지정함으로써 데이터프레임의 인덱스를 T와 F 순서대로 지정하였다. 칼럼명 또한 동일한 방식으로 columns = [i for i in 'TF']로 T와 F 순서대로 지정한다. df\_cm 객체를 출력하면 이전 단계에서 confusion\_matrix()를 이용하여 추출했던 결과가 데이터프레임 형태로 변환된 모습을 확인할 수 있다.

```
plt.figure(figsize=(7,5))
plt.title('confusion matrix without normalization')
sns.heatmap(df_cm, annot=True, fmt='d')
```

[그림 82] 컴퓨전 매트릭스로 히트맵을 그리기 위한 코드



[그림 83] 컴퓨전 매트릭스로 히트맵을 그린 결과

- plt.figure(figsize=(7,5))를 이용하여 히트맵의 크기를 지정한다. plt.title()을 이용하면 원하는 문구로 히트맵의 제목을 표기할 수 있다. 마지막으로 sns.heatmap을 이용하여 df\_cm에 저장한 데이터프레임을 히트맵 형식으로 시각화한다. 예측 빈도가 높거나 낮음에 따라 색으로 표현되는 것을 확인하기 위해 모든 빈도를 정규화하지 않고 그대로 표현하였다. 히트맵의 결과는 confusion\_matrix()의 결과를 해석하는 방식과 동일하며 추가적으로 히트맵에 나타난 색상이 분류 결과의 절대적인 수치(개수)를 구분하기 쉽게 표현되었는지 확인한다.

## [단계 ⑨] 결과 분석 및 해석

### ⑨-1. 분석 결과 해석하기

- 분석 결과를 통해 표준데이터를 이용하여 시간의 흐름에 따라 일정한 패턴을 보이며 변화하는 진동데이터의 불량을 어느 정도 검출할 수 있는지 확인하였다. 표준데이터와 상한, 하한 임계치의 효용성을 확인하기 위해 임계치를 시간의 흐름에 상관 없이 하나의 값으로 설정하였을 때의 불량 검출 비율과 비교할 수 있다.

```

predict = 0
real = 0

for i in range(len(df)):
    if df['real_fault'][i] == 1:
        real += 1
    else:
        real = real

for j in range(len(df)):
    if df['pred_fault'][j] == 1 and df['real_fault'][j] == 1:
        predict += 1
    else:
        predict = predict

print('실제 불량 횟수:', real)
print('표준데이터를 이용한 불량 검출 횟수:', predict)
print('실제 불량 중 표준데이터를 이용하여 불량을 정확히 검출한 비율:', round(recall_score(df['real_fault'], df['pred_fault']), 2)*100, '%')

```

[그림 84] 실제 불량인 경우를 표준데이터를 이용하여 예측하기 위한 코드

```

실제 불량 횟수: 28969
표준데이터를 이용한 불량 검출 횟수: 26707
실제 불량 중 표준데이터를 이용하여 불량을 정확히 검출한 비율: 92.0 %

```

[그림 85] 실제 불량인 경우를 표준데이터를 이용하여 예측한 결과에 대한 정리

- 먼저 분석 결과에서 추출되었던 실제 불량횟수와 표준데이터를 이용한 불량 검출 횟수를 확인하면 위와 같다. [그림 84]의 코드를 이용하여 df 객체의 real\_fault와 pred\_fault 변수에서 실제 불량횟수와 표준데이터를 이용한 불량 검출횟수를 확인

하였다. 또한, 예측 품질(pred\_fault)과 실제 품질(real\_fault)을 비교하여 실제 불량 중 표준데이터를 이용한 불량 검출이 얼마나 잘 되었는지에 대한 재현율을 추출하였다. 앞서 언급했듯이 실제 불량 검출횟수에 대해 표준데이터는 약 92%를 검출해내었다.

```
mean_data = df['real_y'].mean()
print(mean_data)
```

[그림 86] 실제 y값의 평균을 구하기 위한 코드

```
mean_pred = []

for a in range(len(df)):
    if df['real_y'][a] > mean_data:
        mean_pred.append(1)
    else:
        mean_pred.append(0)

df['mean_fault'] = mean_pred
```

[그림 87] 실제 y값의 평균 데이터로 예측 품질을 평가한 칼럼을 추가하기 위한 코드

```
-0.1466728783791479
```

[그림 88] 실제 y값의 평균 계산 결과

- 예를 들어, 임계치를 실제 y값의 평균을 구한 하나의 값이라고 가정한다면 현재 데이터에서 약 -0.15로 계산된다. [그림 86]과 [그림 87]의 코드는 현재 실제 y값에 대한 하나의 평균치를 산출하고 실제 y값이 해당 평균치보다 크면 불량, 작으면 정상이라고 판단하는 mean\_fault 칼럼을 추가한다.

```
predict = 0
real = 0

for i in range(len(df)):
    if df['real_fault'][i] == 1:
        real += 1
    else:
        real = real

for j in range(len(df)):
    if df['mean_fault'][j] == 1 and df['real_fault'][j] == 1:
        predict += 1
    else:
        predict = predict

df['mean_fault'] = mean_pred

print('실제 불량 횟수:', real)
print('평균 임계치보다 클 경우 불량을 검출한 횟수:', predict)
print('실제 불량 중 평균 임계치를 이용하여 불량을 정확히 검출한 비율:',
      round(recall_score(df['real_fault'], df['mean_fault'])*100, 2), '%')
```

[그림 89] 실제 불량인 경우를 평균데이터를 이용하여 예측하기 위한 코드

```

실제 불량 횟수: 28969
평균 임계치보다 클 경우 불량을 검출한 횟수: 13017
실제 불량 중 평균 임계치를 이용하여 불량을 정확히 검출한 비율: 44.93 %
  
```

[그림 90] 실제 y값이 평균 데이터보다 클 경우를 불량으로 하여 예측한 결과에 대한 정리

- 만약 실제 공정에서 위와 같은 하나의 평균값을 임계치로 하여 진동데이터가 임계치보다 클 때 불량으로 검출한다면 평균 임계치를 이용하여 불량 검출이 가능한 비율은 위와 같다. 검출 비율은 약 44.93%로, 표준데이터를 이용했을 때보다 약 47.07%가량 검출 성능이 저하된 것을 확인할 수 있다.

```

mean_pred = []

for a in range(len(df)):
    if df['real_y'][a] < mean_data:
        mean_pred.append(1)
    else:
        mean_pred.append(0)

df['mean_fault'] = mean_pred
  
```

[그림 91] 실제 y값의 평균 데이터로 예측 품질을 평가한 칼럼을 추가하기 위한 코드

```

predict = 0
real = 0

for i in range(len(df)):
    if df['real_fault'][i] == 1:
        real += 1
    else:
        real = real

for j in range(len(df)):
    if df['mean_fault'][j] == 1 and df['real_fault'][j] == 1:
        predict += 1
    else:
        predict = predict

print('실제 불량 횟수:', real)
print('평균 임계치보다 작을 경우 불량을 검출한 횟수:', predict)
print('실제 불량 중 평균 임계치를 이용하여 불량을 정확히 검출한 비율:', round(recall_score(df['real_fault'], df['mean_fault'])*100, 2), '%')
  
```

[그림 92] 실제 불량인 경우를 평균 데이터를 이용하여 예측하기 위한 코드

```

실제 불량 횟수: 28969
평균 임계치보다 작을 경우 불량을 검출한 횟수: 15952
실제 불량 중 평균 임계치를 이용하여 불량을 정확히 검출한 비율: 55.07 %
  
```

[그림 93] 실제 y값이 평균 데이터보다 작을 경우를 불량으로 하여 예측한 결과에 대한 정리

- 또한, 진동데이터가 평균 임계치보다 작을 때 불량으로 검출한다면 평균 임계치를 이용하여 불량 검출이 가능한 비율은 위와 같다. [그림 92]의 코드는 실제 y값이 해당 평균치보다 작으면 불량, 크면 정상이라고 판단하는 mean\_fault 칼럼을 추가한다. 검출 비율은 약 55.07%로, 표준데이터를 이용했을 때보다 약 36.93%가

량 검출 성능이 저하된 것을 확인할 수 있다.

- 평균 임계치보다 크거나 작을 경우 불량을 정확히 검출한 비율은 모두 표준데이터를 사용했을 때보다 낮다. 또한, 92%의 재현율에 더하여 위에서 언급했던 정밀도는 80%, f1-score는 85%로 모두 평균 임계치를 사용했을 때보다 좋은 분류 성능을 산출했으며, 정상인 경우와 불량인 경우에 대한 분류 성능을 통틀어 간주하는 정확도는 87%로 산출되었다. 즉, 표준데이터는 정상인 경우와 불량인 경우를 구분하지 않고 모두 일정 수준 이상 정확히 검출해낼 수 있음을 확인하였다.
- 이와 같이 K-최근접 이웃 회귀를 이용하여 표준데이터와 상한, 하한 임계치를 추출하고 이를 기준으로 공정의 불량을 검출하면 시간의 흐름과 센서데이터의 패턴에 따라 유연한 적용이 가능하다.

#### ⑨-2. 제조현장 관점에서 분석결과 해석하기

- 상기 분석 결과를 통해, K-최근접 이웃 회귀 알고리즘을 현장에 적용하여 용해탱크 교반구동장치의 예지보전을 위한 설비 이상징후 예측을 효과적으로 수행할 수 있음을 알 수 있다. 본 가이드북의 분석을 통해 도출한 상한, 하한 임계치 값을 기준으로 한 이상 검출 성능은 상대적으로 낮았지만, 표준데이터를 기준으로 한 이상 검출은 약 92%로, 현장에 적용할 수 있을 정도의 정확성이 있음을 확인하였다.
- 본 가이드북에서 개발한 분석 모델의 현장 적용은 공정중 교반구동기에 부착된 진동센서를 통해 실시간 시계열 진동데이터가 수집되고, 본 분석을 통해 도출한 표준데이터와 실시간으로 비교하여 이상징후 발견시 알람을 발생하도록 하여, 실제 설비 문제가 발생하기 전에 선제적으로 대응할 수 있도록 한다.
- 하지만 위와 같이 실시간 예지보전을 수행하기 위해서는 실시간 스트리밍 분석 인공지능 추론 플랫폼을 먼저 구축하여야 한다. 이 플랫폼에 해당 인공지능 모델과 파라미터를 탑재하고, 실시간 시계열 진동데이터를 입력값(input)으로 넣을 시, 실시간으로 이상징후를 판단할 수 있다. 물론, 이와 같이 실시간 시계열 데이터와 추론 플랫폼을 활용하기 위해서는 실시간 데이터 수집 및 관리 플랫폼(스마트공장) 구축이 수반되어야 한다.

### 3. 유사 타 현장의 「교반구동장치 AI 데이터셋」 분석 적용

#### 3.1 본 분석이 적용 가능한 제조현장 소개

- 모터 구동장치가 있는 설비를 사용하는 대부분의 중소기업 제조업 체에서는 본 가이드북에서 제시된 것과 동일한 문제를 안고 있을 것이다. 이와 같은 어려움이 있는 산업현장에서 만약 센서를 통해 설비 진동데이터 수집이 가능하고, 실제 결함여부 데이터를 전산화하여 관리하고 있다면 본 분석은 적용 가능하다.

#### 3.2 본 「교반구동장치 AI 데이터셋」 분석을 원용하여 타 제조현장 적용 시, 주요 고려사항

- 설비/모터에 따라 분석 결과가 달라지기 때문에 해당 모터의 데이터로 분석 모델을 다시 개발하여야 한다.
- 설비/모터 상태를 나타내는 다른 데이터(초음파, 열화상, 전류, 온도 등)들도 포함하여 모델을 개발 및 평가할 경우 더 정확한 예측모델을 생성할 수도 있다.
- 본 분석에서 활용한 특성값은 타 설비와 맞지 않을 수 있기 때문에 현장 전문가의 의견을 반영하여 적용 여부를 결정해야 한다.

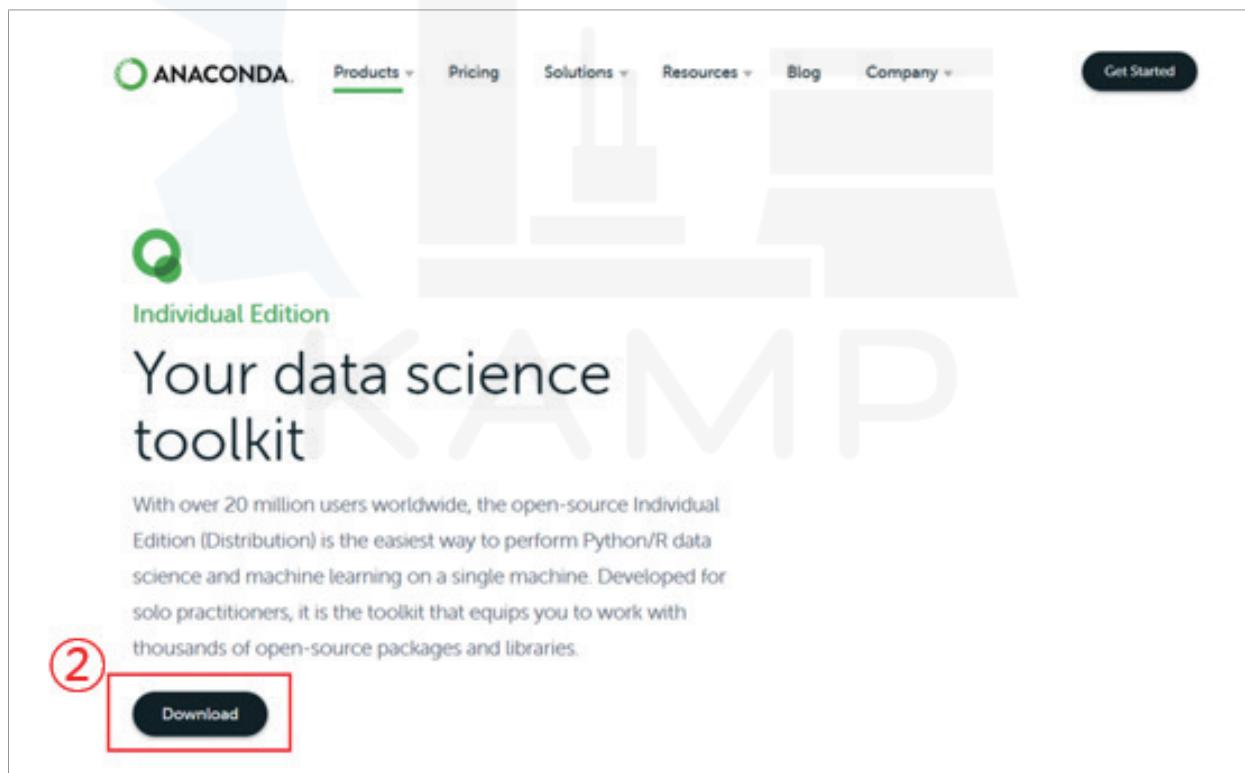
### ◎ Python을 사용하는 이유

- 데이터를 분석 및 예측하기 위해 Python 프로그래밍 언어를 사용하는 이유는 다음과 같다. 첫째, 비교적 읽고 쓰기 쉬운 프로그래밍 언어이며, 둘째, 효율적인 메모리 관리 기능을 갖추고 있고, 셋째, 머신러닝 프레임워크와 라이브러리가 풍부하다는 장점이 있다.

### ◎ Python 플랫폼 : Anaconda

- 먼저 Python 플랫폼인 Anaconda를 설치한다. Anaconda는 핵심적인 과학 및 수학 Python 라이브러리를 포함한 패키지이며, 머신러닝에 필수적인 툴이다. 쉽게 말해서, Anaconda를 설치하면 분석과 예측을 수행하기 위해 필요한 다양한 패키지들을 쉽게 설치할 수 있으며, 코드를 직접 입력하고 실행시킬 ‘개발환경’을 Anaconda Navigator라는 하나의 통합된 공간에서 편리하게 설치할 수 있다. 분석에 들어가기에 앞서 다음 순서에 맞춰 Anaconda를 설치한다.

### ◎ Anaconda Installer 설치



## Anaconda Installers

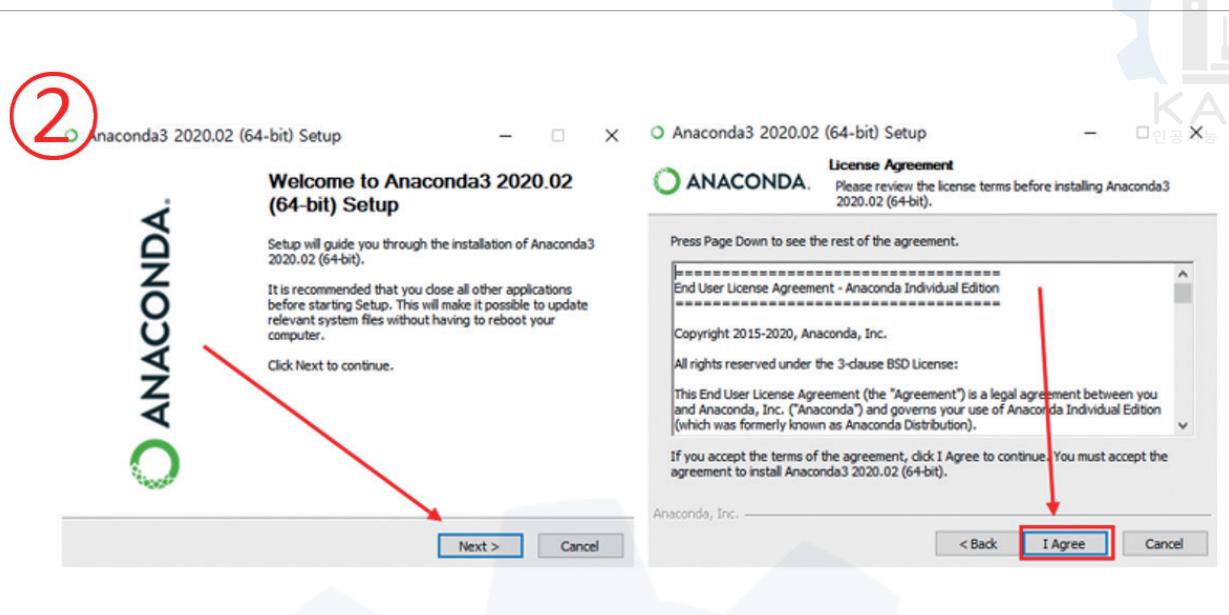
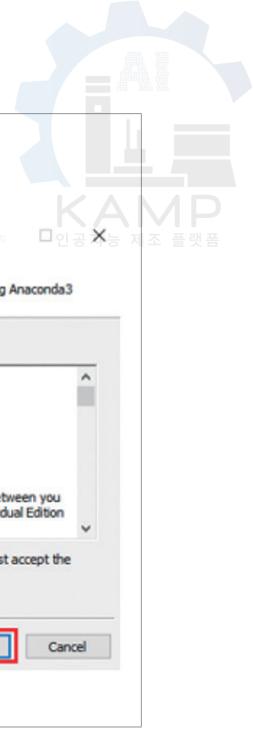
(3)

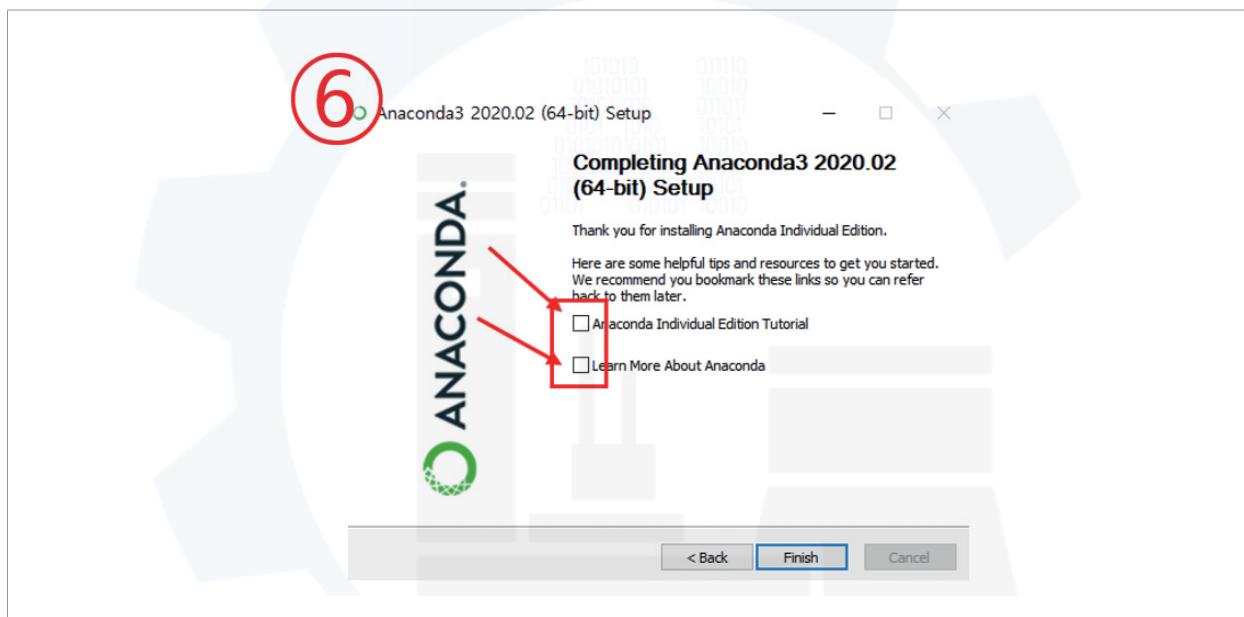
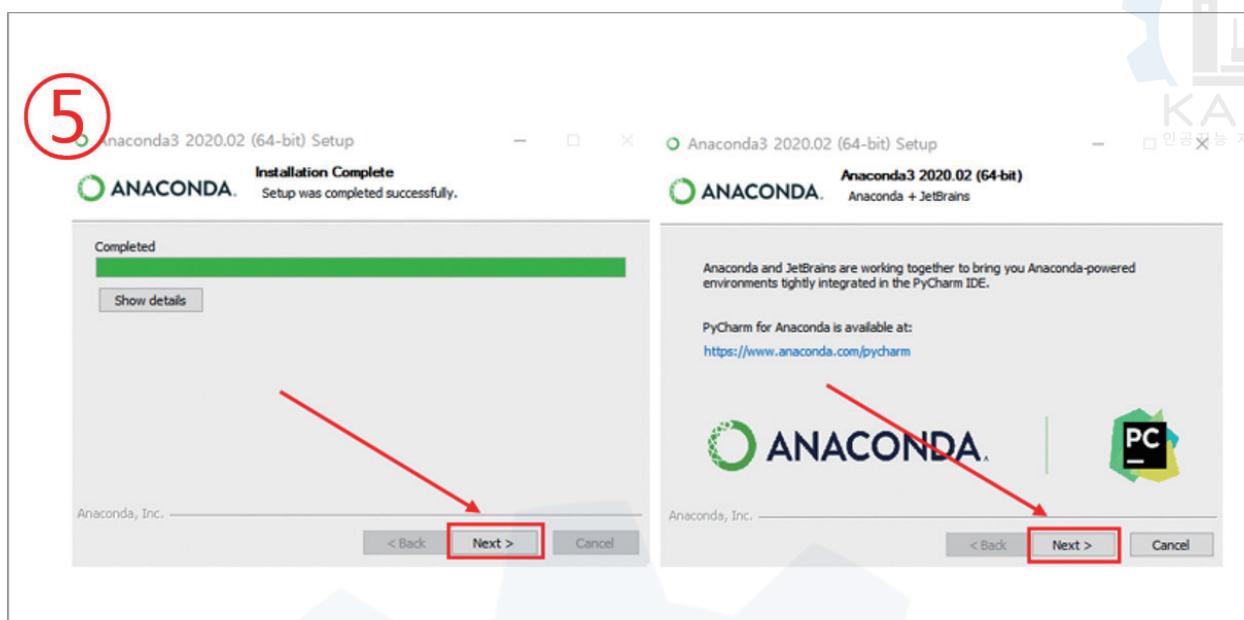
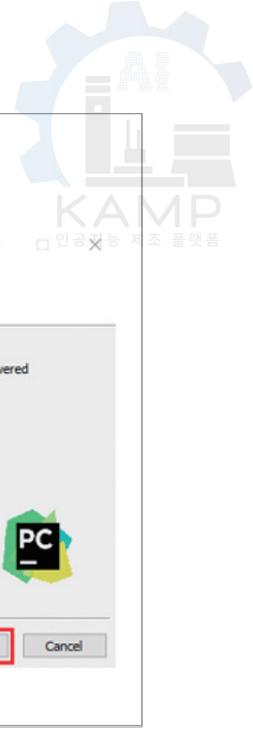
Windows	MacOS	Linux
Python 3.8 64-Bit Graphical Installer (466 MB)	Python 3.8 64-Bit Graphical Installer (462 MB)	Python 3.8 64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

- ① [www.anaconda.com/download](http://www.anaconda.com/download) 주소로 접속한다.
- ② 'Download' 버튼을 클릭한다.
- ③ Windows, MacOS, Linux 버전 중 자신의 PC 운영체제와 일치하는 버전을 선택하여 Installer를 다운로드 한다.

### ◉ Anaconda 실행파일 설치(Windows 64-bit 기준)

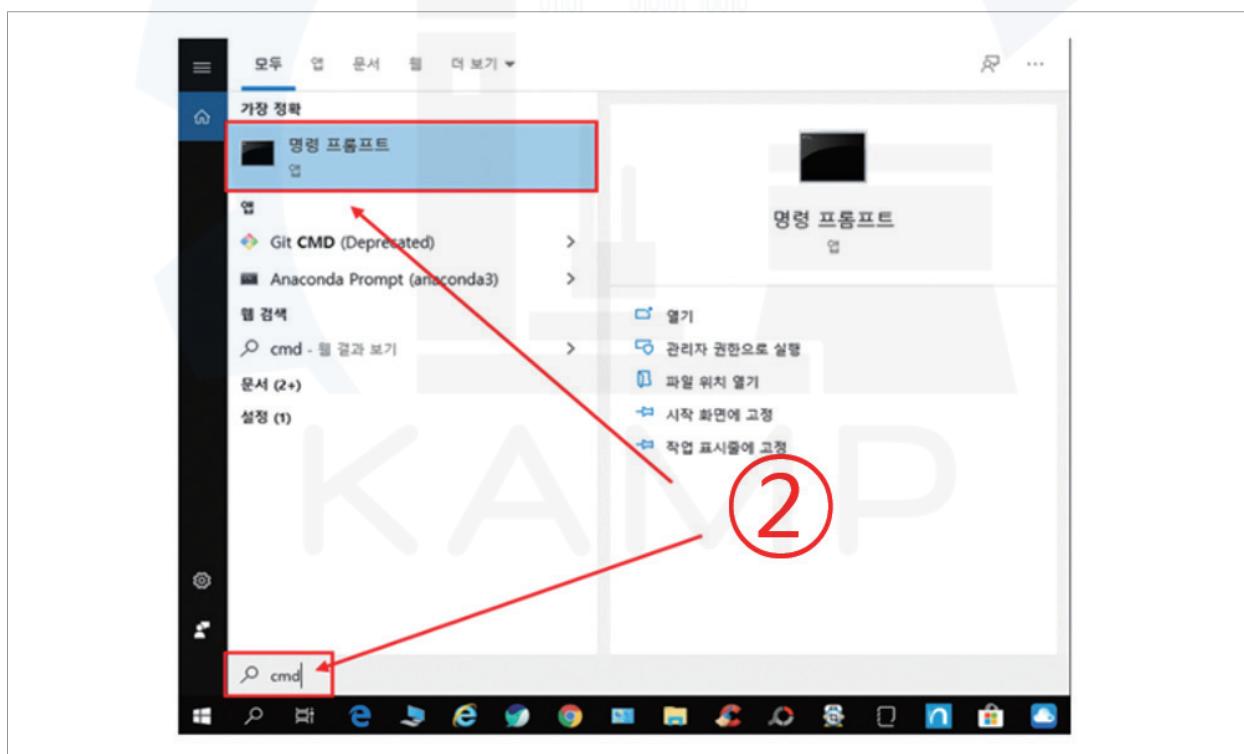
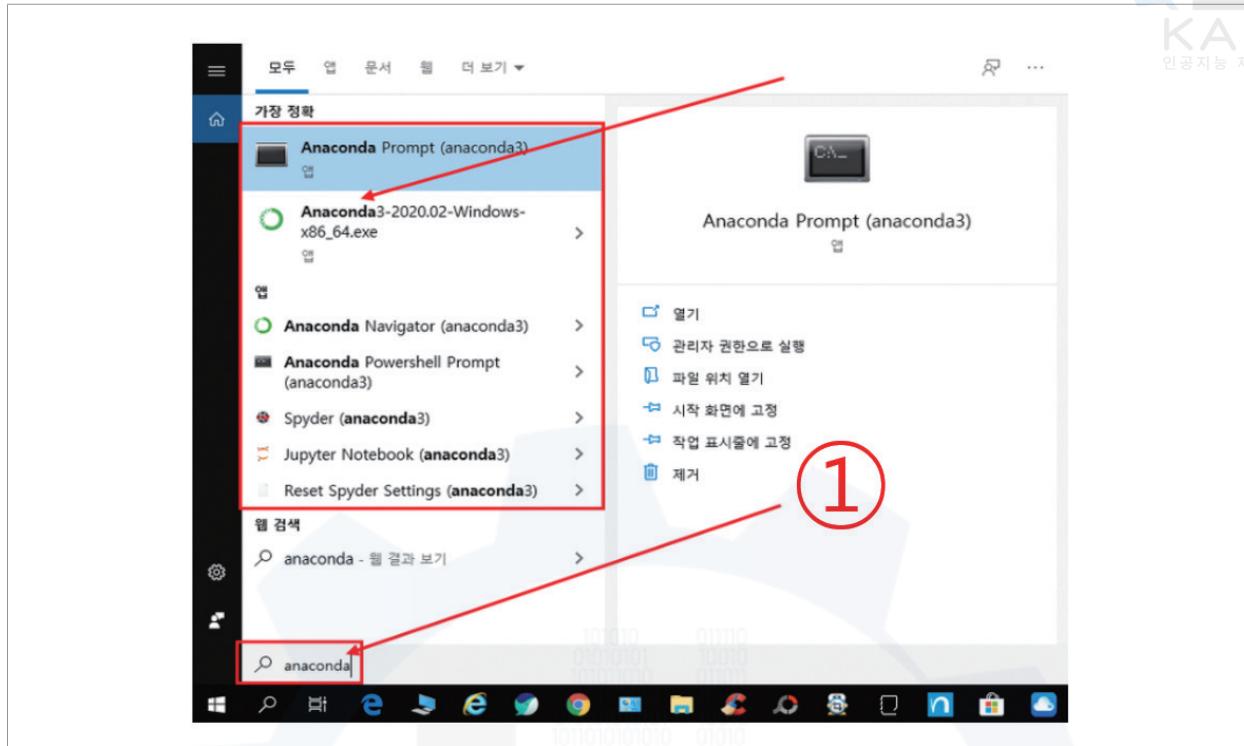




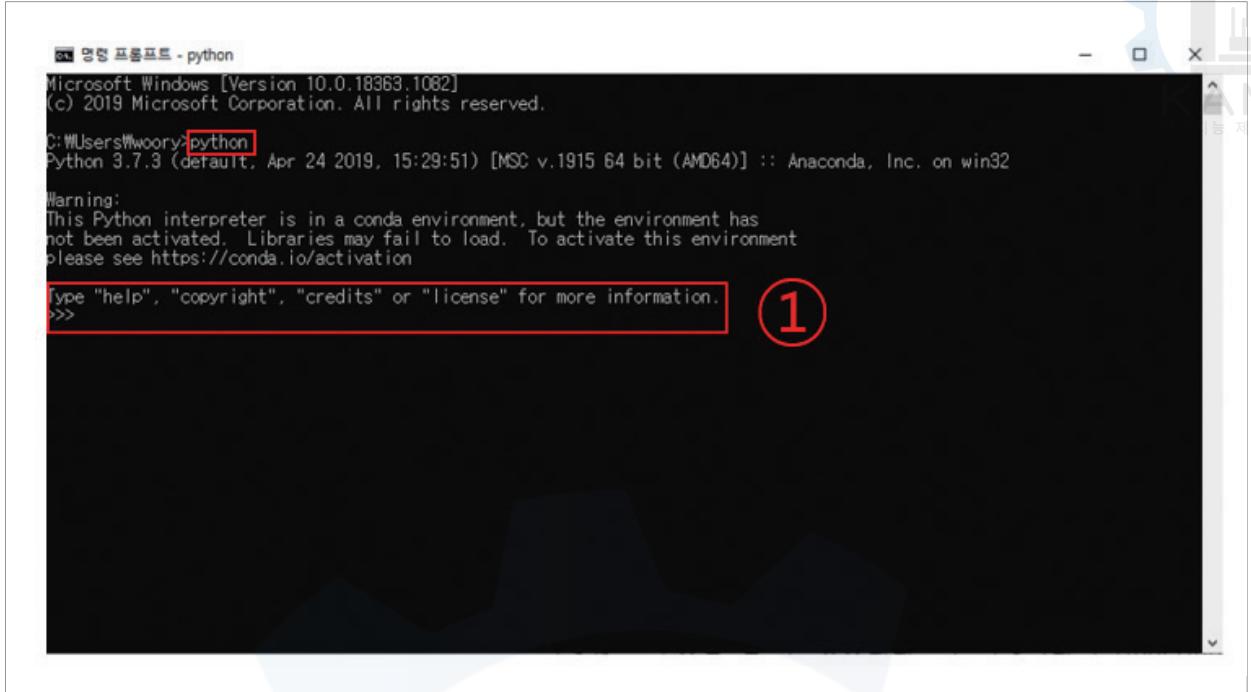


- ① 설치된 실행 파일을 열 때는 [마우스 오른쪽]-[관리자 권한으로 실행]을 클릭한다.
- ② 별도의 설정 없이 Next 버튼을 눌러 다음 단계로 진행한다.
- ③ 'Just me(recommended)'를 체크하고 넘어가서 다운로드 경로는 별도로 설정하지 않고 바로 넘어간다.
- ④ 체크박스를 모두 체크한 후 Install을 클릭한다.
- ⑤ 별도의 설정 없이 Next 버튼을 눌러 진행한다.
- ⑥ 두 가지 체크박스는 Anaconda와 관련된 튜토리얼과 교육자료에 관한 것을 설치 및 연결할 것인지 묻는 창이므로 체크를 전부 해제하고 Finish를 클릭한다.

## ① Anaconda 설치 확인



- ① 설치가 완료되면 Windows 좌측 하단의 돋보기를 클릭하여 Anaconda를 입력한 후 설치된 것을 확인한다.
- ② Anaconda를 통해 Python이 잘 설치되었는지 최종적으로 확인하기 위해 'cmd'를 입력한 후 '명령 프롬프트' 앱을 클릭한다.

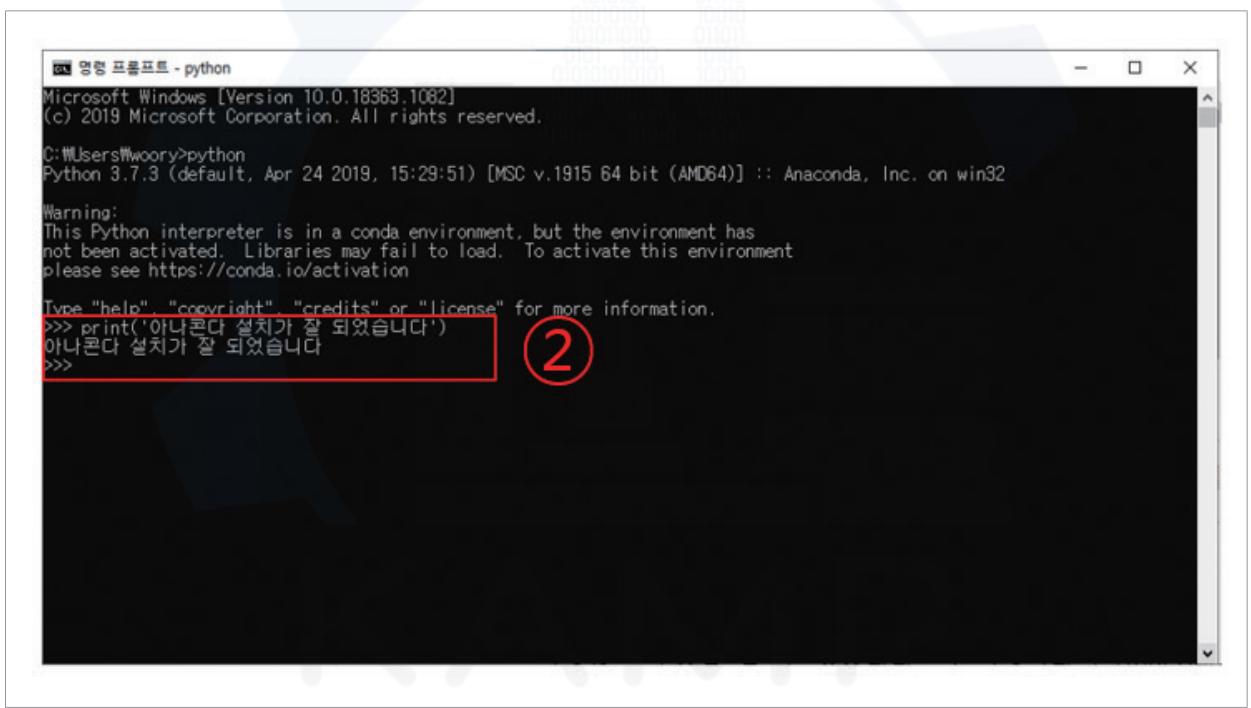


```
C:\Users\woory>python
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\woory>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>>
```



```
C:\Users\woory>python
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\woory>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

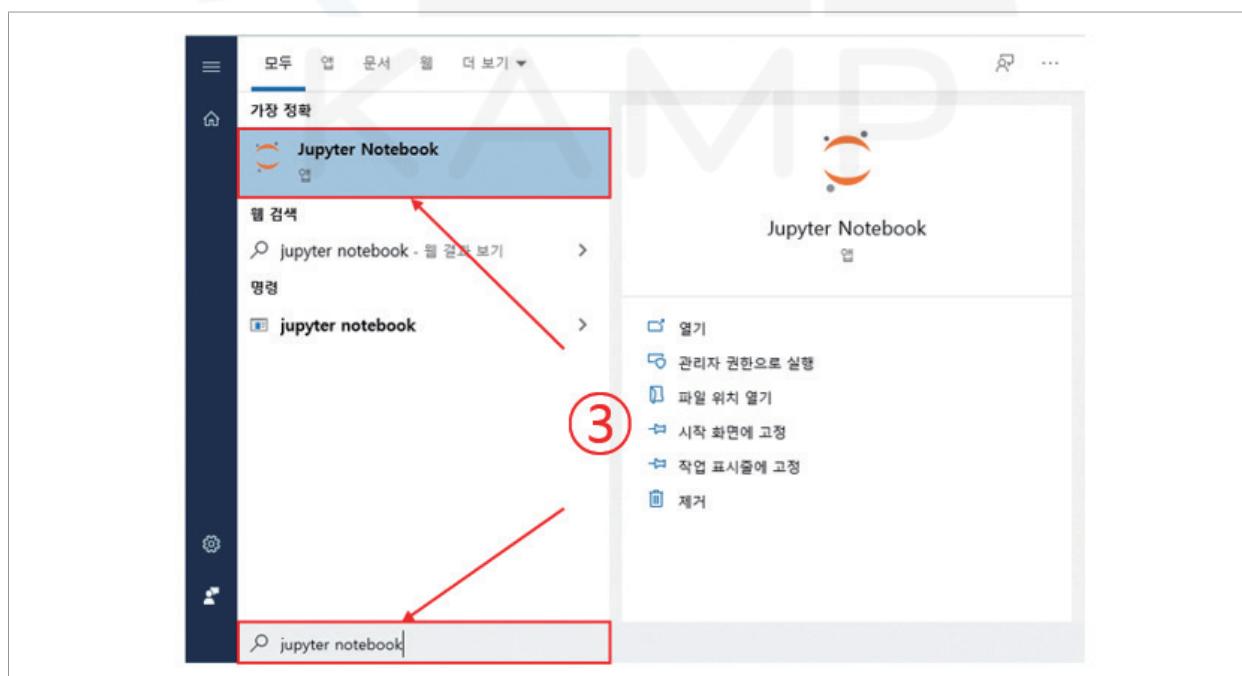
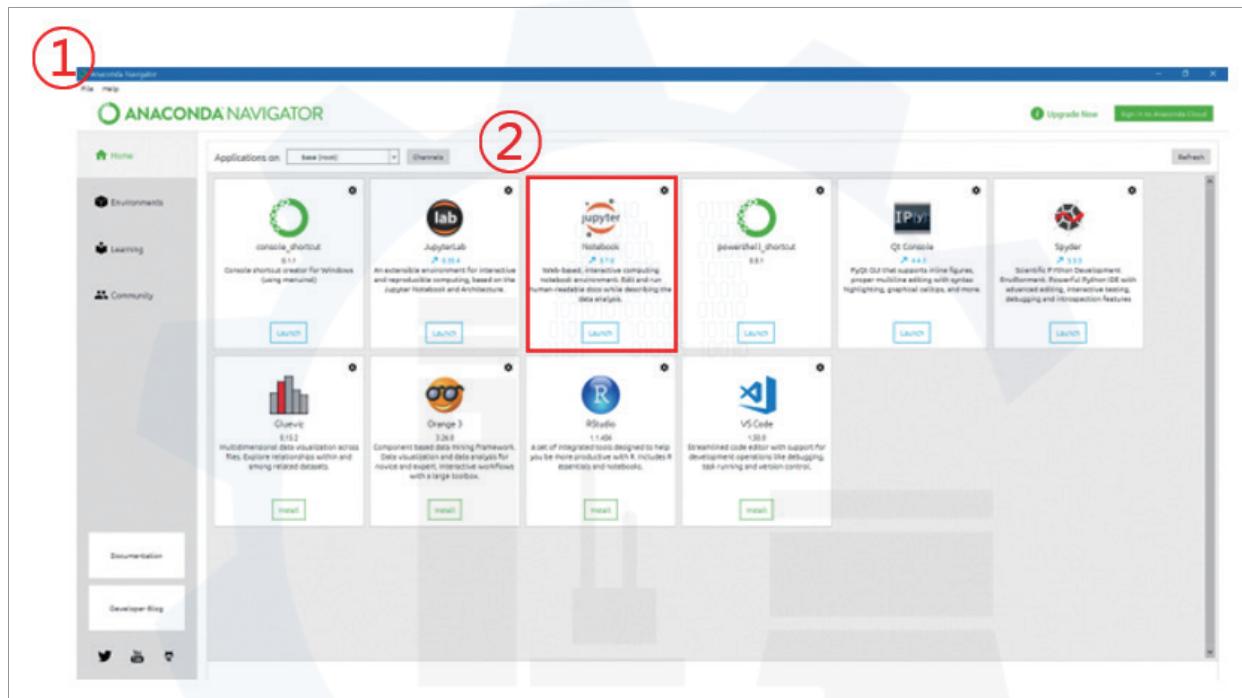
Type "help", "copyright", "credits" or "license" for more information.
>>> print('아나콘다 설치가 잘 되었습니다.')
아나콘다 설치가 잘 되었습니다.
>>>
```

- ① 'python'을 입력한 후 Enter를 눌러 Python이 실행되는 모습을 확인한다.
- ② 최종적으로 코드 실행이 정상적으로 작동하는지 확인하기 위해 '>>>' 옆에 'print('아나콘다 설치가 잘 되었습니다.')'를 입력하고 Enter를 눌러 해당 문자열이 그대로 화면에 출력되는 모습을 확인한다.

## ◎ Python 개발환경 : Jupyter Notebook

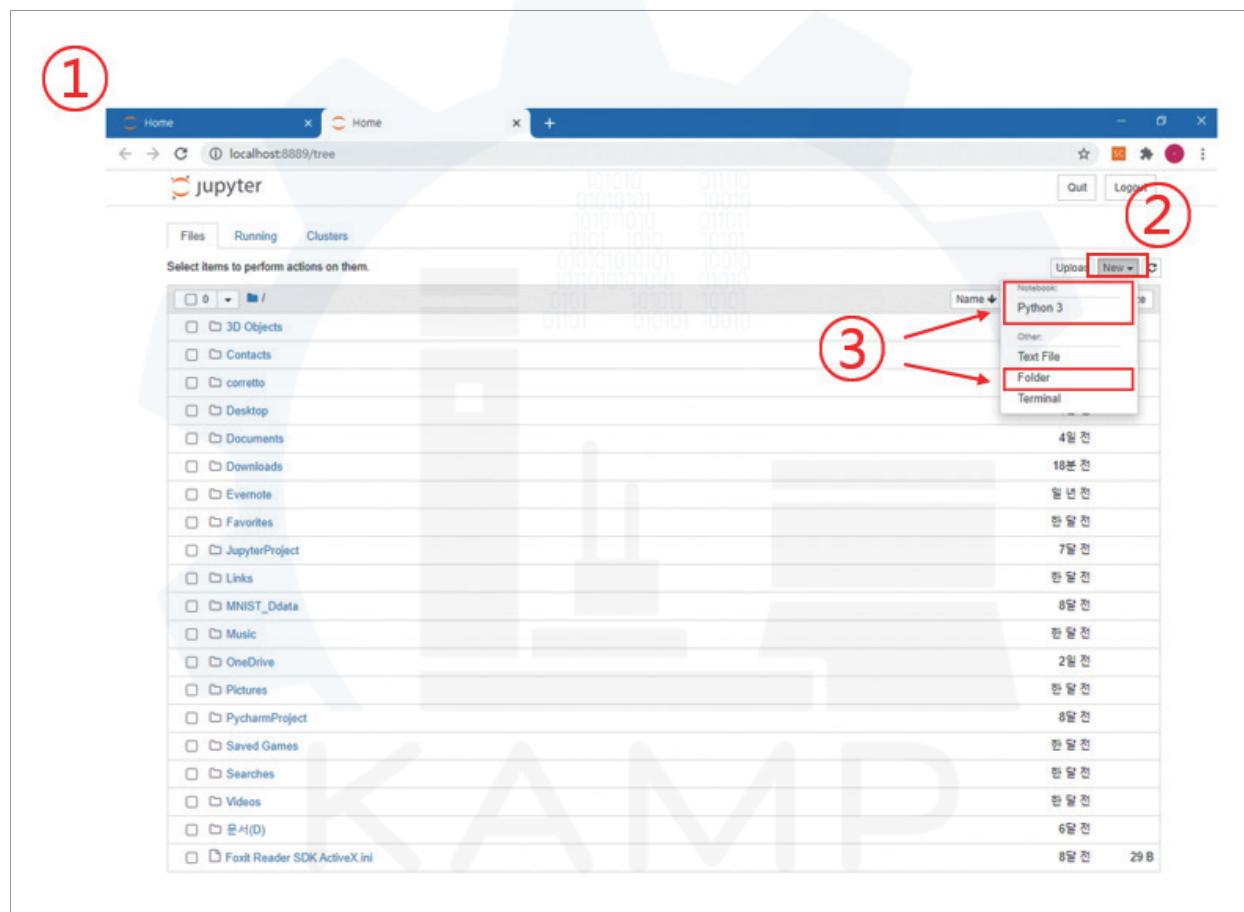
- Jupyter Notebook이란 ‘컴퓨터에게 Python 언어로 명령을 내렸을 때 즉각적인 응답이 오는 대화형 개발환경’이다. 여기서 말하는 ‘Notebook’은 일종의 ‘브라우저 웹페이지’ 형태이다. 즉, Jupyter Notebook이라는 웹페이지 상에서 Python 코드를 한 줄씩 입력하여 시행하고 분석 결과를 얻는다.

## ◎ Jupyter Notebook 설치



- ① Anaconda Navigator를 검색 후 실행한다.
- ② Jupyter Notebook이 'Launch' 상태인지 확인한다.  
 (※ 'Install' 상태일 경우, 클릭하여 설치해야 한다.)
- ③ Windows 화면 좌측 하단의 돋보기 버튼을 눌러 'jupyter notebook'을 입력하고 실행 앱이 있는지 확인한다.  
 (※ Anaconda Navigator를 매번 실행하지 않아도 Jupyter Notebook 앱을 통해 곧바로 실행이 가능하다.)

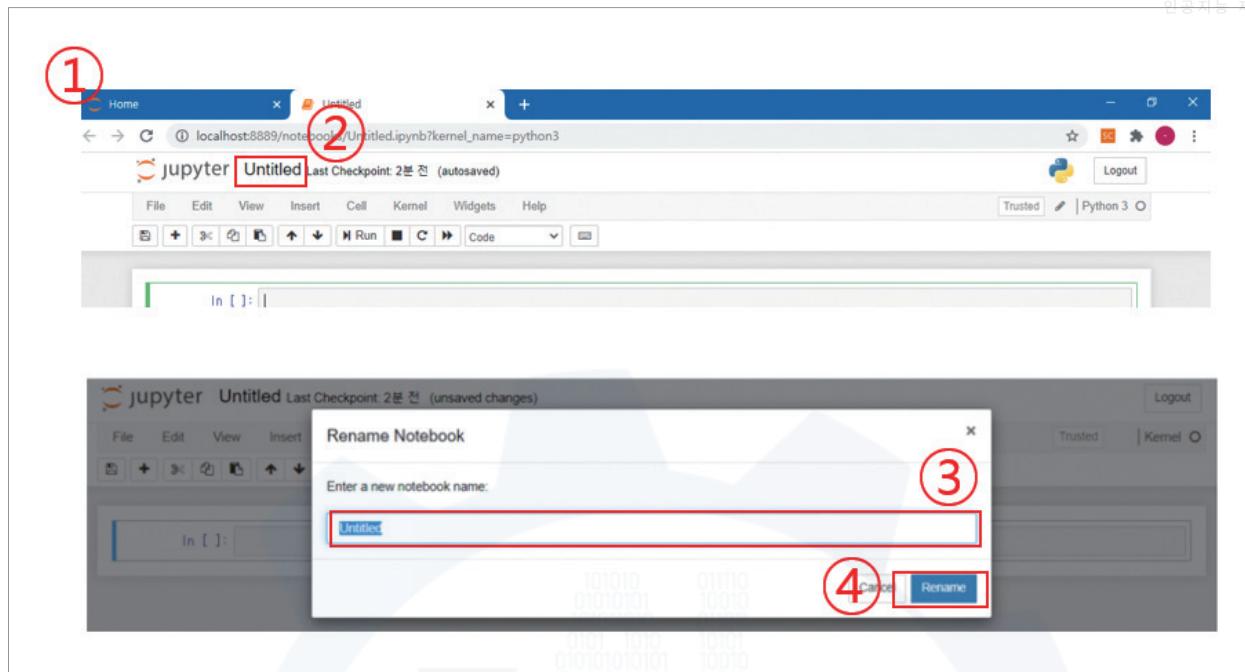
## ◉ Jupyter Notebook 실행 후 Notebook 생성



- ① Jupyter Notebook 앱을 클릭하여 웹페이지가 실행되는 것을 확인한다.
- ② notebook을 하나를 생성하기 위해 'New' 버튼을 클릭한다.
- ③ 현재 위치에 코드 파일 notebook을 바로 생성하려면 'Python 3'을 클릭하고, 새로운 폴더를 생성 후 그 안에 생성하려면 'Folder'를 클릭한다.  
 (※ 새로운 폴더 생성 후 진행할 경우, 그 폴더 안에서 [New]-[Python 3]를 동일하게 클릭한다.)

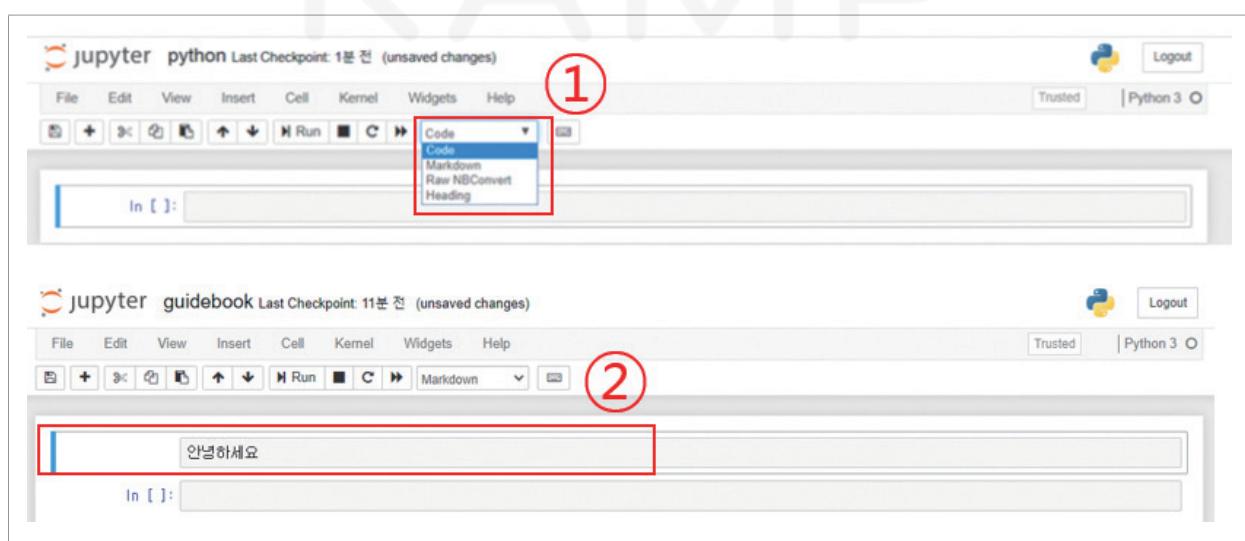
## ◉ Notebook 기본 활용법 : 제목 설정 및 주석 사용

- 먼저 제목을 설정할 경우 다음과 같은 과정을 거친다.



- ① 새롭게 생성된 notebook을 연다.
- ② 처음 생성한 notebook의 제목을 확인하고 수정할 경우 제목을 두 번 클릭한다.
- ③ Rename Notebook 창에 수정하고자 하는 notebook의 제목을 입력한다.
- ④ 수정을 완료하기 위해 'Rename' 버튼을 클릭한다.

- 다음으로 코드 내에 주석(설명)을 기입할 경우 다음과 같은 과정을 거친다. ‘마크다운(Markdown)’을 이용하여 코드 실행 줄을 메모장 형식으로 변형하거나 이러한 변형 과정 없이 코드를 입력한 실행 줄 내에 곧바로 기입할 수 있다.



```
jupyter guidebook Last Checkpoint: 11분 전 (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 Logout
In [1]: 안녕하세요
NameError
<ipython-input-1-041a565a82b3> in <module>
      ----> 1 안녕하세요
NameError: name '안녕하세요' is not defined
```

```
jupyter guidebook Last Checkpoint: 13분 전 (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 Logout
In [1]: 안녕하세요
NameError
<ipython-input-1-041a565a82b3> in <module>
      ----> 1 안녕하세요
NameError: name '안녕하세요' is not defined
In [2]: #안녕하세요
In [ ]:
```

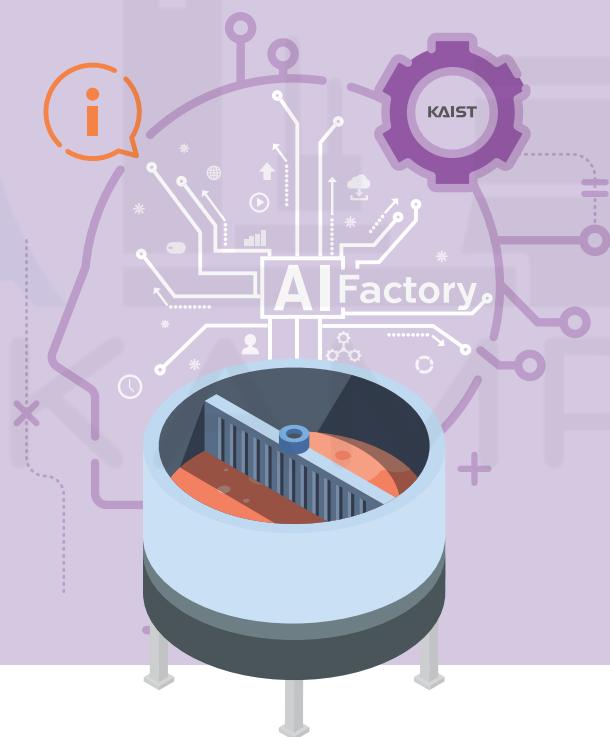
- ① 마크다운을 사용하기 위해 코드 실행 줄 하나를 선택해두고 ‘Code’로 설정되어 있는 버튼을 클릭하여 ‘Markdown’을 선택한다.
- ② 메모장 형식으로 코드 실행 줄이 변경된 것을 확인하고 주석 내용을 입력하여 Shift+Enter 키를 눌러 주석을 설정한다.
- ③ 마크다운으로 변경하지 않은 코드 실행 줄에 주석을 입력할 경우 오류가 발생하는 것을 확인 할 수 있다.
- ④ 마크다운 변경 없이 주석을 기입할 경우 '#'을 먼저 입력한 후 그 뒤에 설명을 기입한다.

## ◉ Notebook 기본 활용법 : 단축키

- 자주 활용하는 단축키는 해당 코드 실행 줄을 클릭해놓은 상태에서 활용한다.

- ① Shift + Enter : 해당 코드 줄을 실행(Run)한다.
- ② m : 해당 코드 줄을 마크다운으로 자동 변경한다.
- ③ y : 해당 마크다운 줄을 코드 줄로 자동 변경한다.
- ④ a : 바로 위에 코드 줄을 추가한다.
- ⑤ b : 바로 아래에 코드 줄을 추가한다.
- ⑥ x : 해당 코드 줄을 삭제한다.

# 「교반구동장치 AI 데이터셋」 분석실습 가이드북



중소벤처기업부



스마트제조혁신추진단



34141 대전광역시 유성구 대학로 291 한국과학기술원(KAIST)  
T. (042)350-2114 F. (042)350-2210(2220)