온도, 녹조, 더러움…

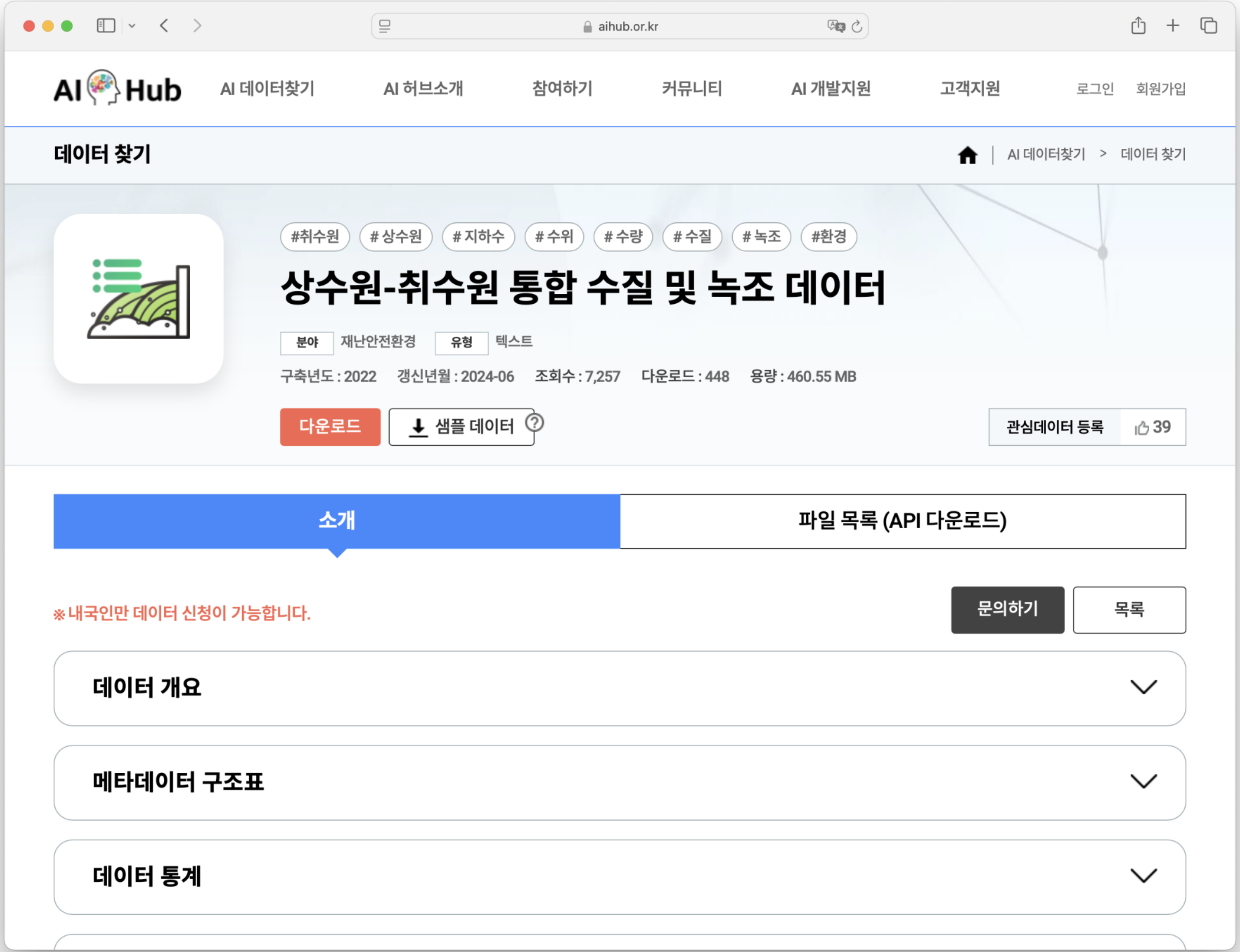
팔당댐 수질관리 데이터 분석

~회귀 문제의 분류 문제화와 함께~

2025/04/27

박종현

공과대학 컴퓨터정보통신공학과



데이터 찾기

#취수원 # 상수원 # 지하수 # 수위 # 수량 # 수질 # 녹조 #환경

상수원-취수원 통합 수질 및 녹조 데이터

분야	재난안전환경
유형	텍스트

구축년도 : 2022 갱신년월 : 2024-06 조회수 : 7,257 다운로드 : 448 용량 : 460.55 MB

※ 내국인만 데이터 신청이 가능합니다.

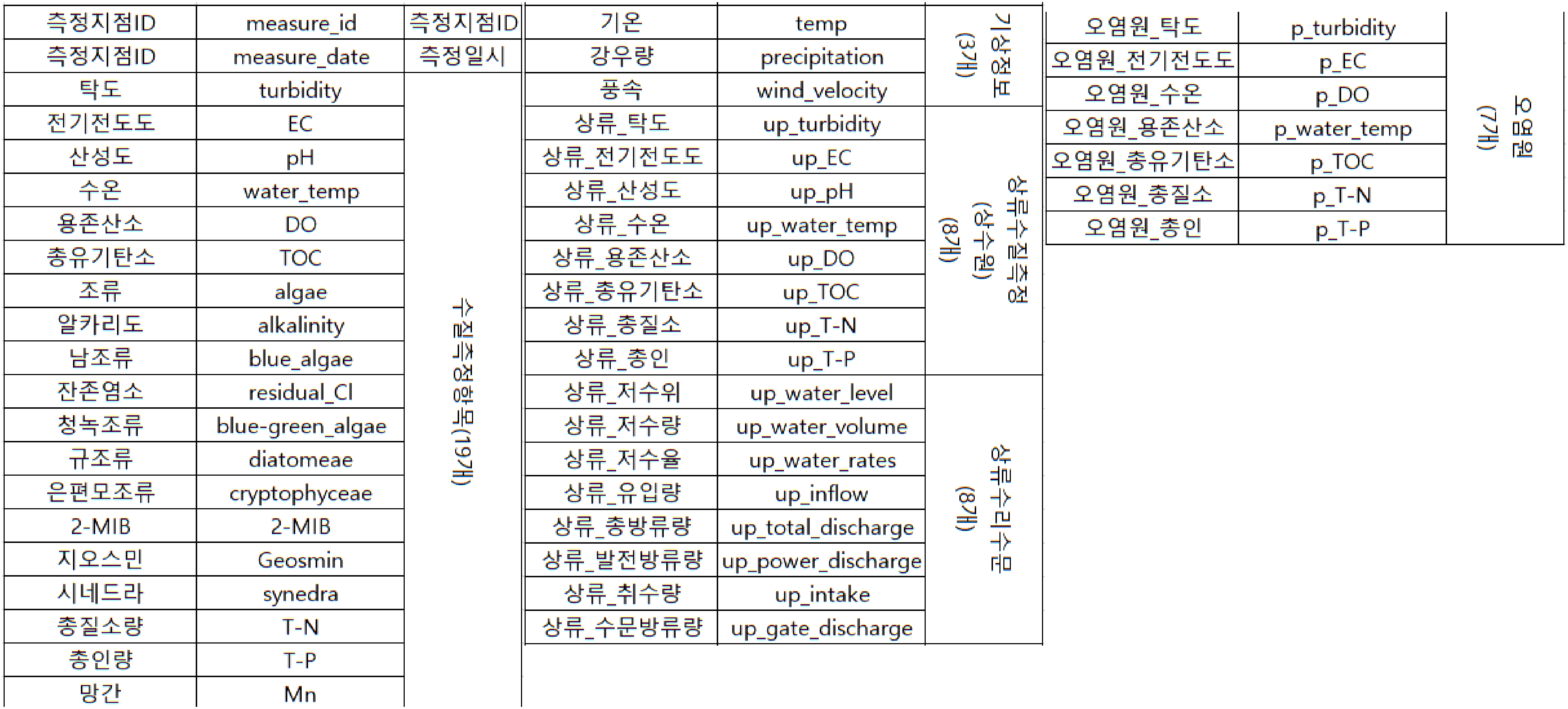
데이터 개요

메타데이터 구조표

데이터 통계

온도와 녹조의 상관관계

녹조 데이터는 사진으로 제공되고 있었음



측정지점ID	measure_id	측정지점ID
측정지점ID	measure_date	측정일시
탁도	turbidity	수질측정항목(19개)
전기전도도	EC	
산성도	pH	
수온	water_temp	
용존산소	DO	
총유기탄소	TOC	
조류	algae	
알카리도	alkalinity	
남조류	blue_algae	
잔존염소	residual_Cl	
청녹조류	blue-green_algae	
규조류	diatomeae	
은편모조류	cryptophyceae	
2-MIB	2-MIB	
지오스민	Geosmin	
시네드라	synedra	
총질소량	T-N	
총인량	T-P	
망간	Mn	

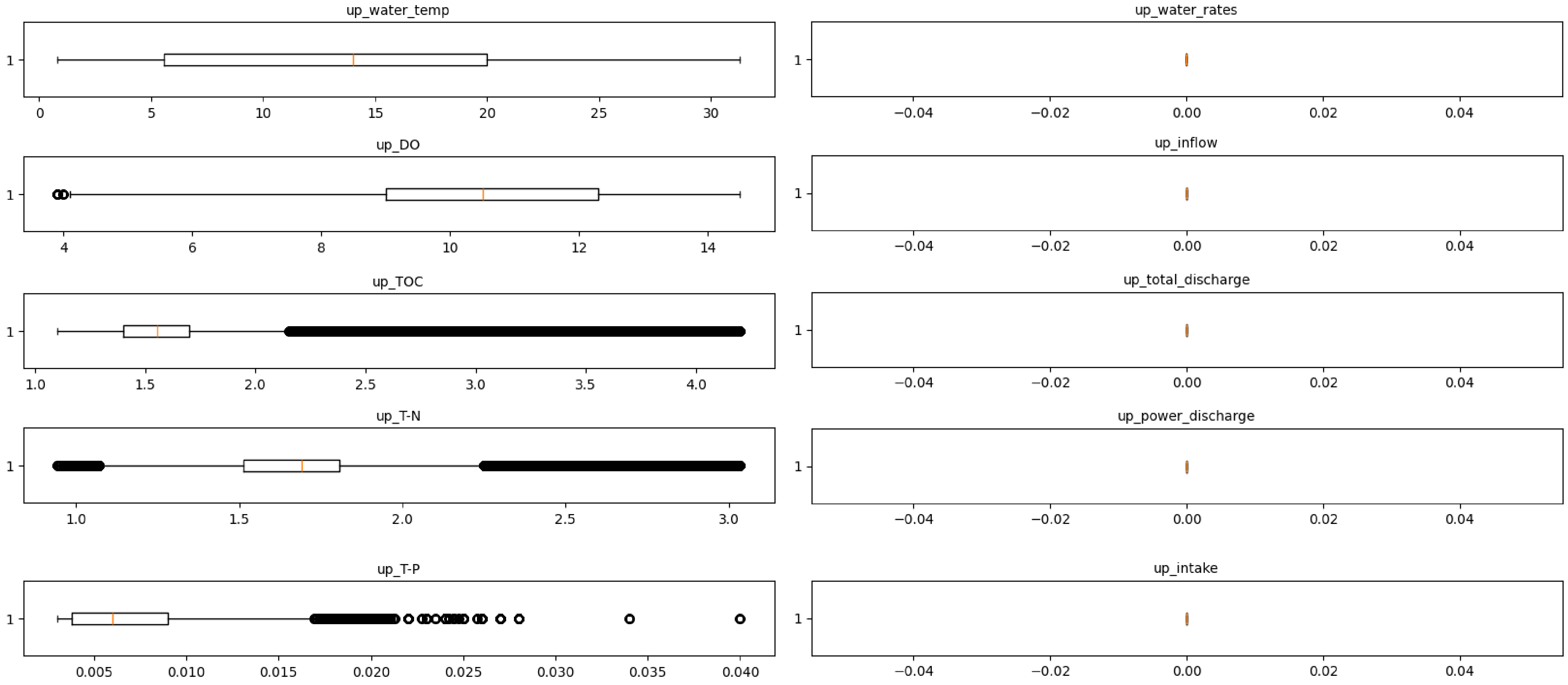
기온	temp	기상정보 (3개)
강우량	precipitation	
풍속	wind_velocity	
상류_탁도	up_turbidity	상류수질측정 (상수원) (8개)
상류_전기전도도	up_EC	
상류_산성도	up_pH	
상류_수온	up_water_temp	
상류_용존산소	up_DO	
상류_총유기탄소	up_TOC	
상류_총질소	up_T-N	
상류_총인	up_T-P	
상류_저수위	up_water_level	상류수리수문 (8개)
상류_저수량	up_water_volume	
상류_저수율	up_water_rates	
상류_유입량	up_inflow	
상류_총방류량	up_total_discharge	
상류_발전방류량	up_power_discharge	
상류_취수량	up_intake	
상류_수문방류량	up_gate_discharge	

오염원_탁도	p_turbidity	오염원 (7개)
오염원_전기전도도	p_EC	
오염원_수온	p_DO	
오염원_용존산소	p_water_temp	
오염원_총유기탄소	p_TOC	
오염원_총질소	p_T-N	
오염원_총인	p_T-P	

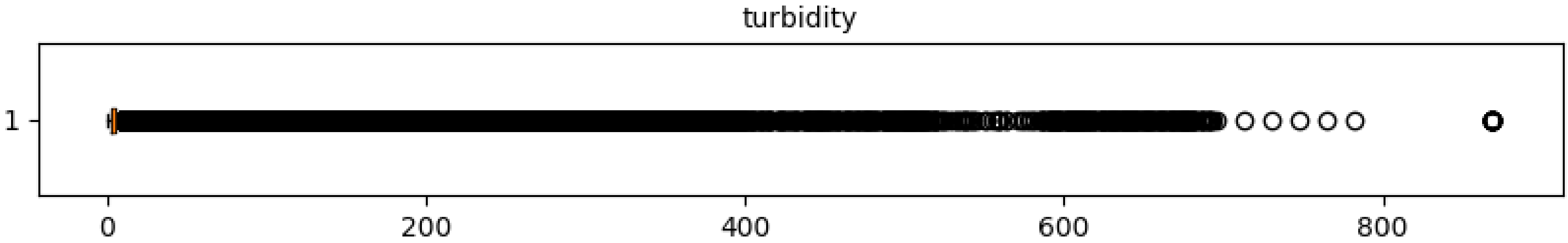
데이터는 온전한가

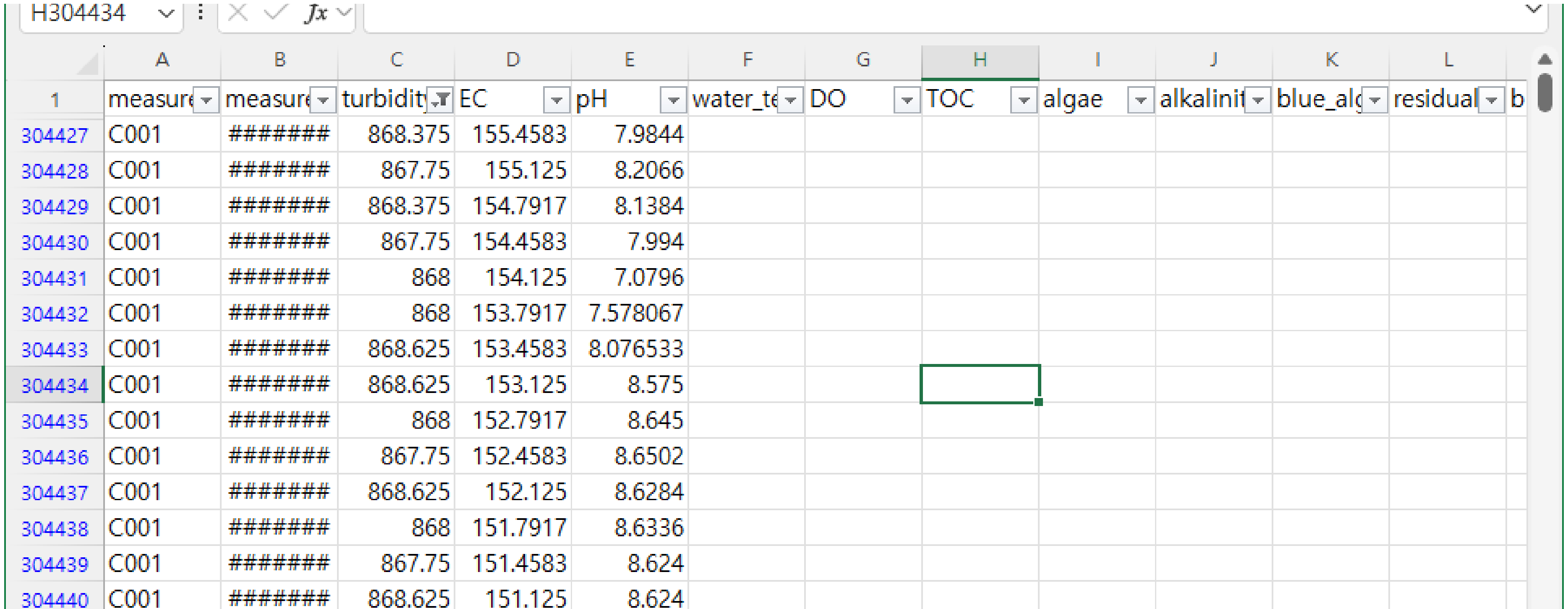
#	Column	Count	Non-Null	Dtype
0	measure_id	1312920	non-null	object
1	measure_date	1312920	non-null	object
2	turbidity	1312920	non-null	float64
3	EC	1312920	non-null	float64
4	pH	1312920	non-null	float64
5	water_temp	0	non-null	float64
6	DO	0	non-null	float64
7	TOC	0	non-null	float64
8	algae	0	non-null	float64
9	alkalinity	0	non-null	float64
10	blue_algae	0	non-null	float64
11	residual_Cl	0	non-null	float64

ax.boxplot(df[each])



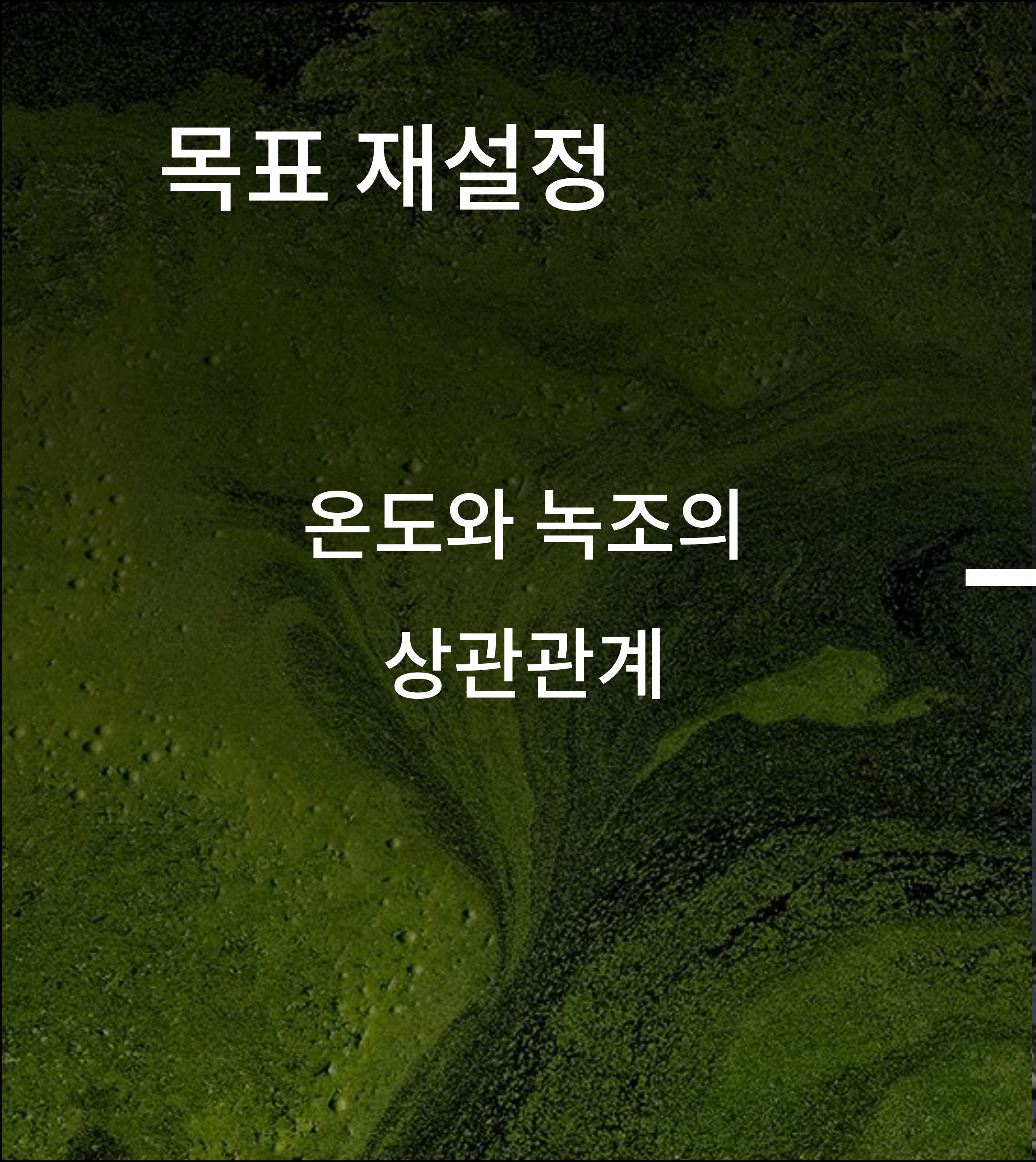
언급할만한 데이터





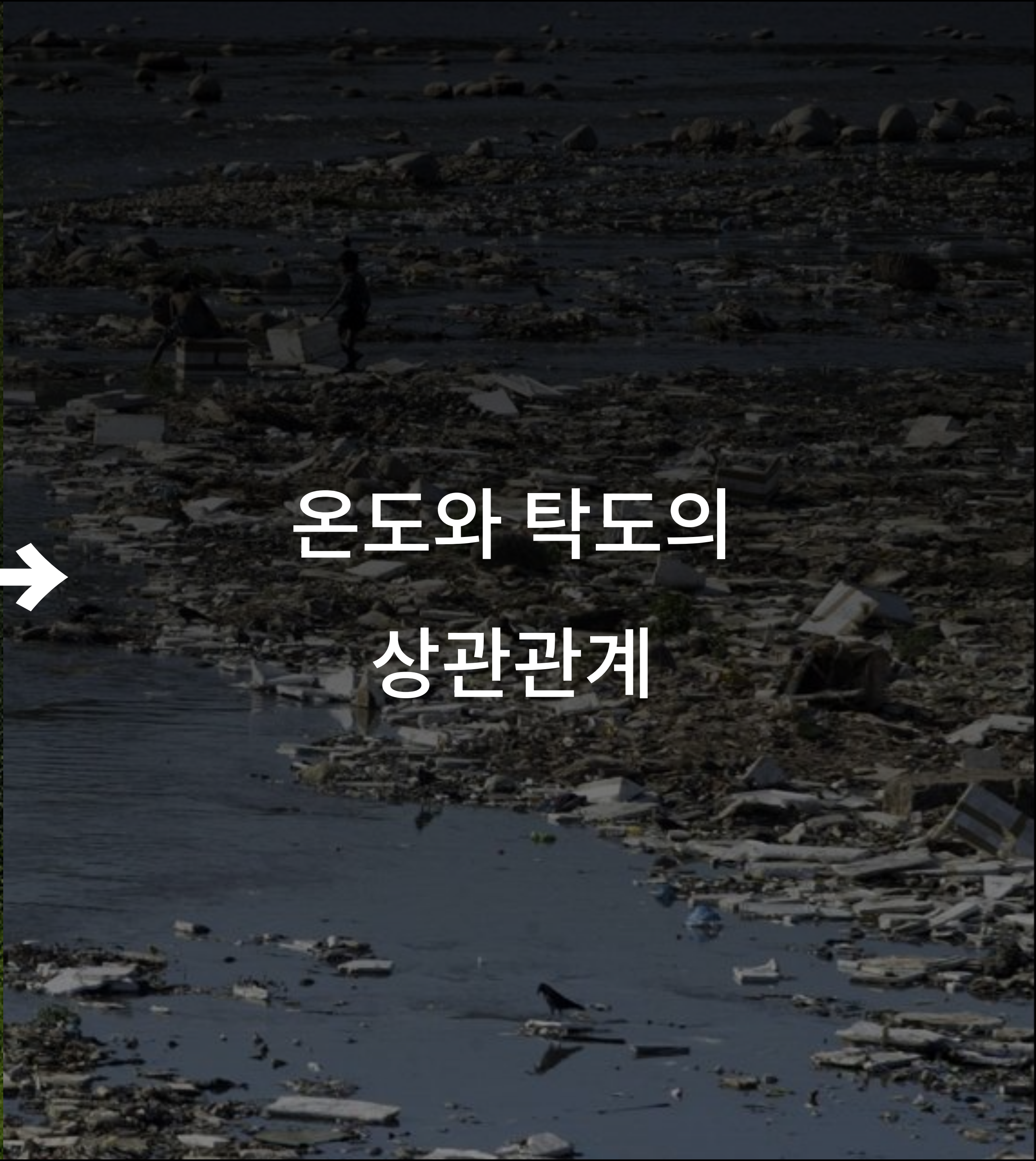
	A	B	C	D	E	F	G	H	I	J	K	L	
1	measure	measure	turbidity	EC	pH	water_te	DO	TOC	algae	alkalinit	blue_alg	residual	b
304427	C001	#######	868.375	155.4583	7.9844								
304428	C001	#######	867.75	155.125	8.2066								
304429	C001	#######	868.375	154.7917	8.1384								
304430	C001	#######	867.75	154.4583	7.994								
304431	C001	#######	868	154.125	7.0796								
304432	C001	#######	868	153.7917	7.578067								
304433	C001	#######	868.625	153.4583	8.076533								
304434	C001	#######	868.625	153.125	8.575								
304435	C001	#######	868	152.7917	8.645								
304436	C001	#######	867.75	152.4583	8.6502								
304437	C001	#######	868.625	152.125	8.6284								
304438	C001	#######	868	151.7917	8.6336								
304439	C001	#######	867.75	151.4583	8.624								
304440	C001	#######	868.625	151.125	8.624								

목표 재설정



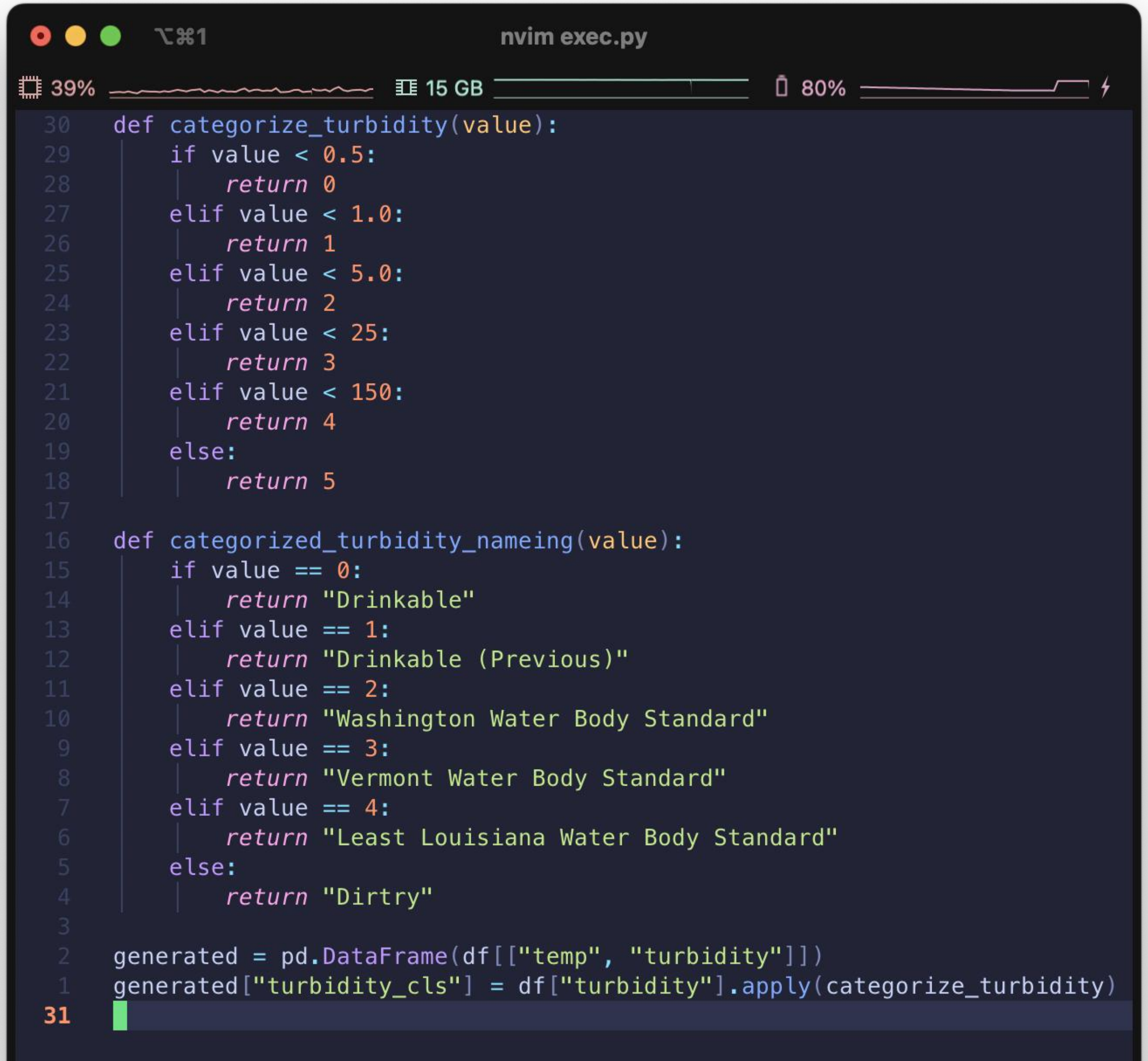
온도와 녹조의 상관관계

→

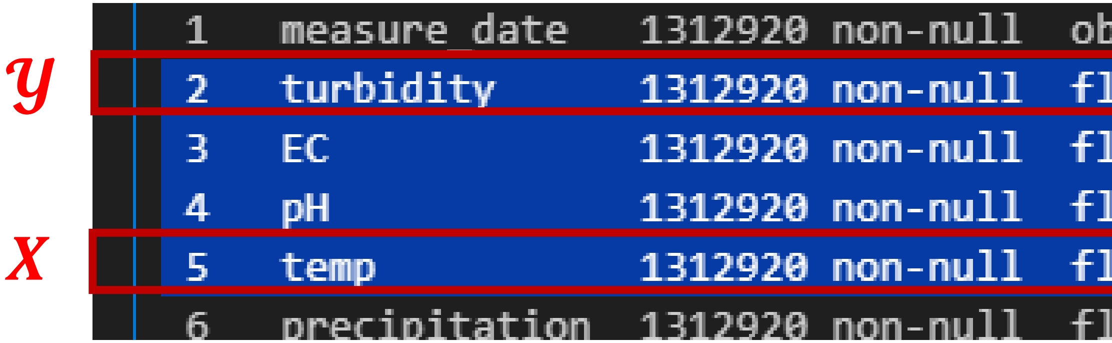


온도와 탁도의 상관관계

회귀 문제의 분류 문제화

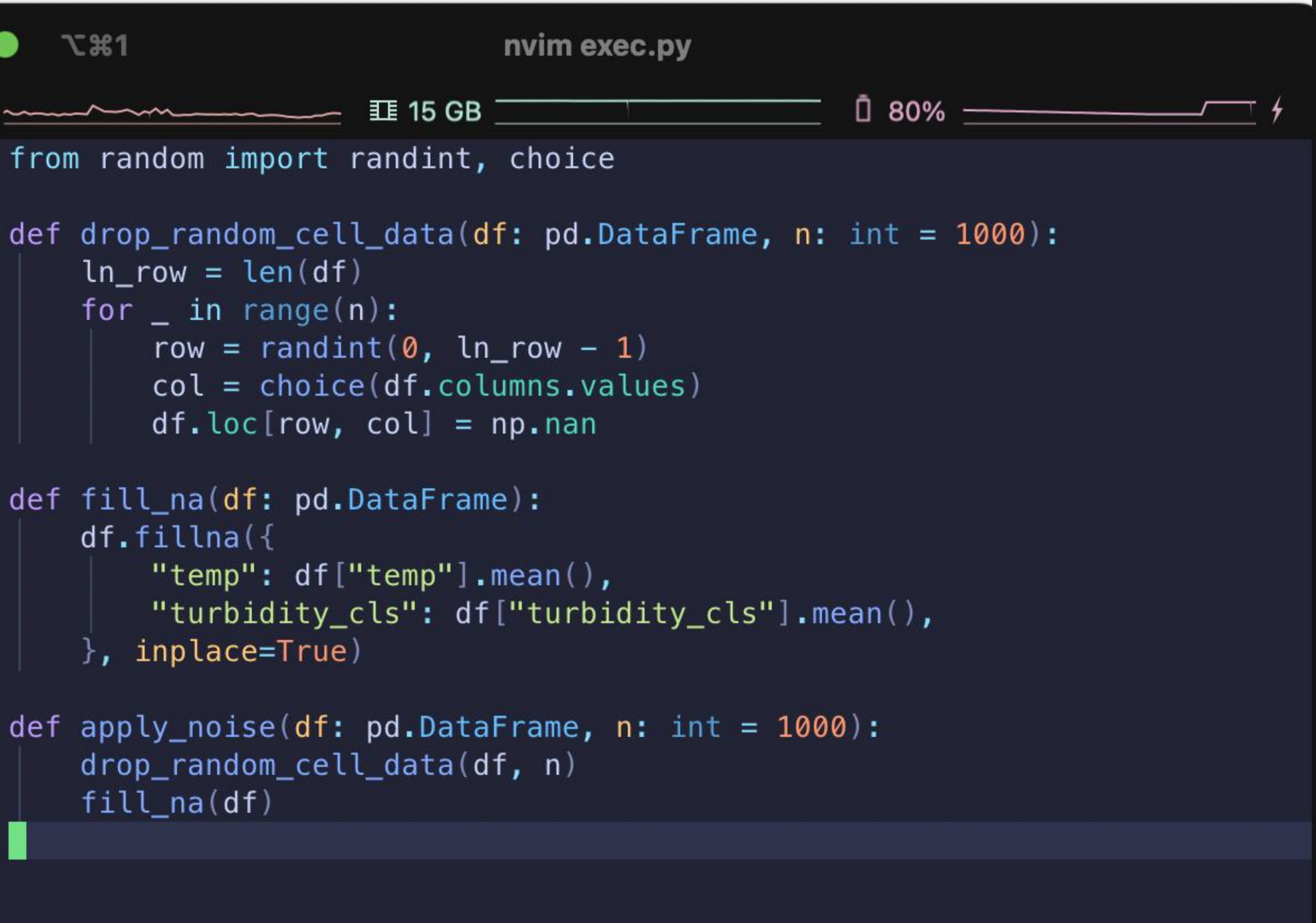


```python
def categorize_turbidity(value):
    if value < 0.5:
        return 0
    elif value < 1.0:
        return 1
    elif value < 5.0:
        return 2
    elif value < 25:
        return 3
    elif value < 150:
        return 4
    else:
        return 5

def categorized_turbidity_nameing(value):
    if value == 0:
        return "Drinkable"
    elif value == 1:
        return "Drinkable (Previous)"
    elif value == 2:
        return "Washington Water Body Standard"
    elif value == 3:
        return "Vermont Water Body Standard"
    elif value == 4:
        return "Least Louisiana Water Body Standard"
    else:
        return "Dirtry"

generated = pd.DataFrame(df[["temp", "turbidity"]])
generated["turbidity_cls"] = df["turbidity"].apply(categorize_turbidity)
```

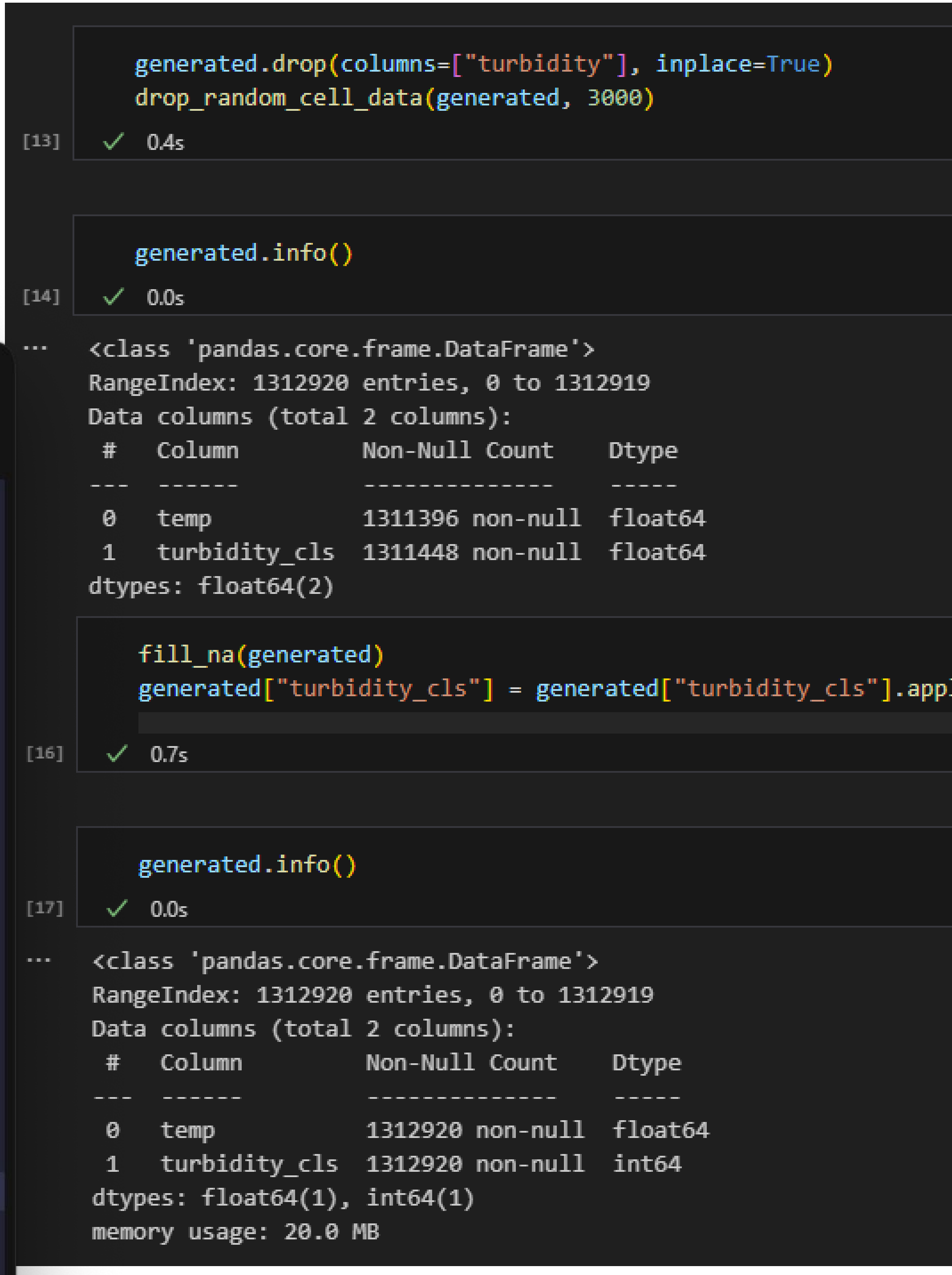


```
 1   measure_date   1312920 non-null   ob
 2   turbidity      1312920 non-null   fl
 3   EC             1312920 non-null   fl
 4   pH             1312920 non-null   fl
 5   temp           1312920 non-null   fl
 6   precipitation  1312920 non-null   fl
```

노이즈 생성

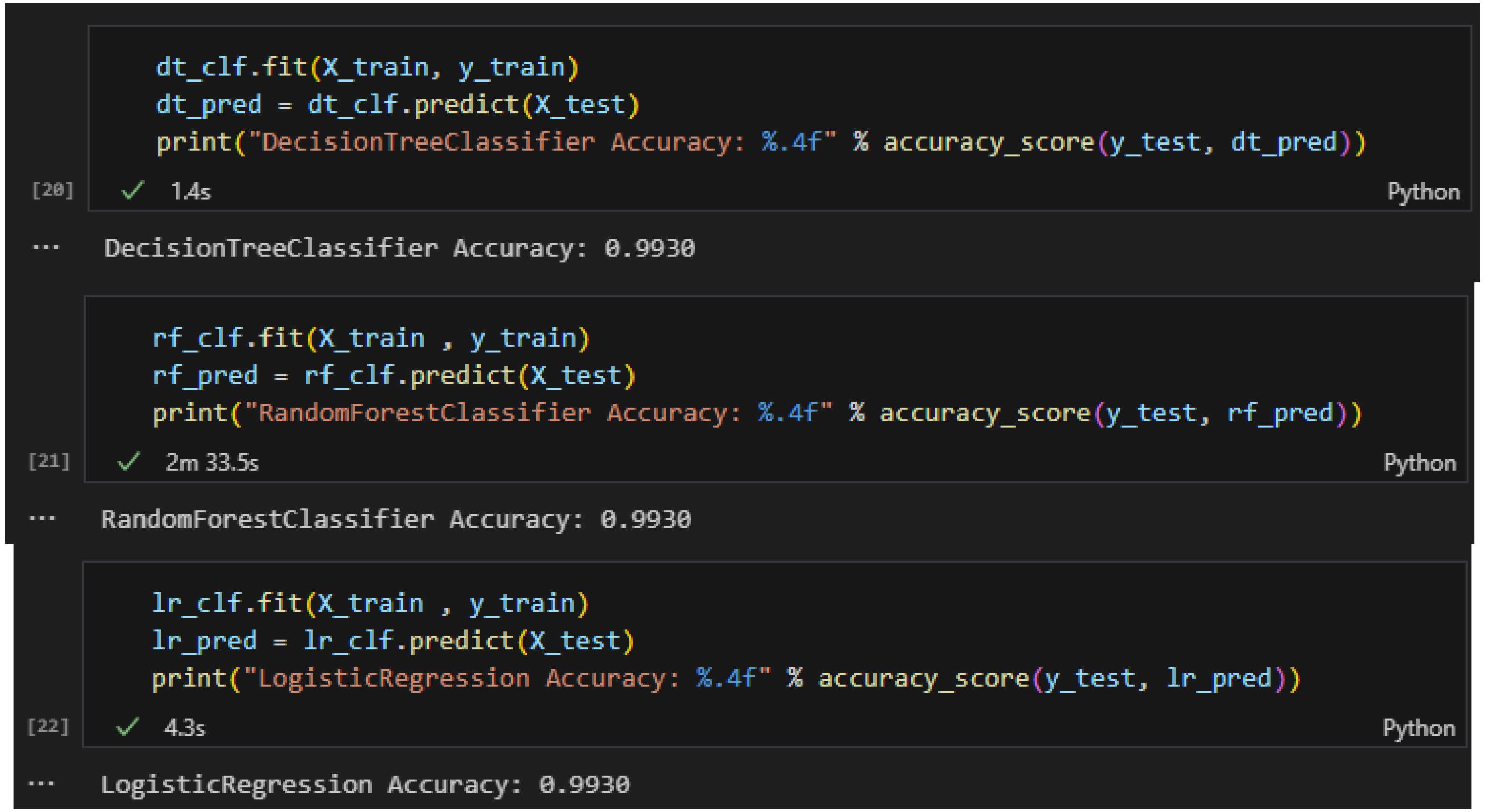


```python
from random import randint, choice

def drop_random_cell_data(df: pd.DataFrame, n: int = 1000):
    ln_row = len(df)
    for _ in range(n):
        row = randint(0, ln_row - 1)
        col = choice(df.columns.values)
        df.loc[row, col] = np.nan

def fill_na(df: pd.DataFrame):
    df.fillna({
        "temp": df["temp"].mean(),
        "turbidity_cls": df["turbidity_cls"].mean(),
    }, inplace=True)

def apply_noise(df: pd.DataFrame, n: int = 1000):
    drop_random_cell_data(df, n)
    fill_na(df)
```



```
[13]
generated.drop(columns=["turbidity"], inplace=True)
drop_random_cell_data(generated, 3000)
✓ 0.4s

[14]
generated.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1312920 entries, 0 to 1312919
Data columns (total 2 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   temp           1311396 non-null  float64
 1   turbidity_cls  1311448 non-null  float64
dtypes: float64(2)

[16]
fill_na(generated)
generated["turbidity_cls"] = generated["turbidity_cls"].appl
✓ 0.7s

[17]
generated.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1312920 entries, 0 to 1312919
Data columns (total 2 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   temp           1312920 non-null  float64
 1   turbidity_cls  1312920 non-null  int64
dtypes: float64(1), int64(1)
memory usage: 20.0 MB
```

분류기 피팅 및 정확도 평가



```python
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
print("DecisionTreeClassifier Accuracy: %.4f" % accuracy_score(y_test, dt_pred))
```

[20] ✓ 1.4s Python

```
DecisionTreeClassifier Accuracy: 0.9930
```

```python
rf_clf.fit(X_train , y_train)
rf_pred = rf_clf.predict(X_test)
print("RandomForestClassifier Accuracy: %.4f" % accuracy_score(y_test, rf_pred))
```

[21] ✓ 2m 33.5s Python

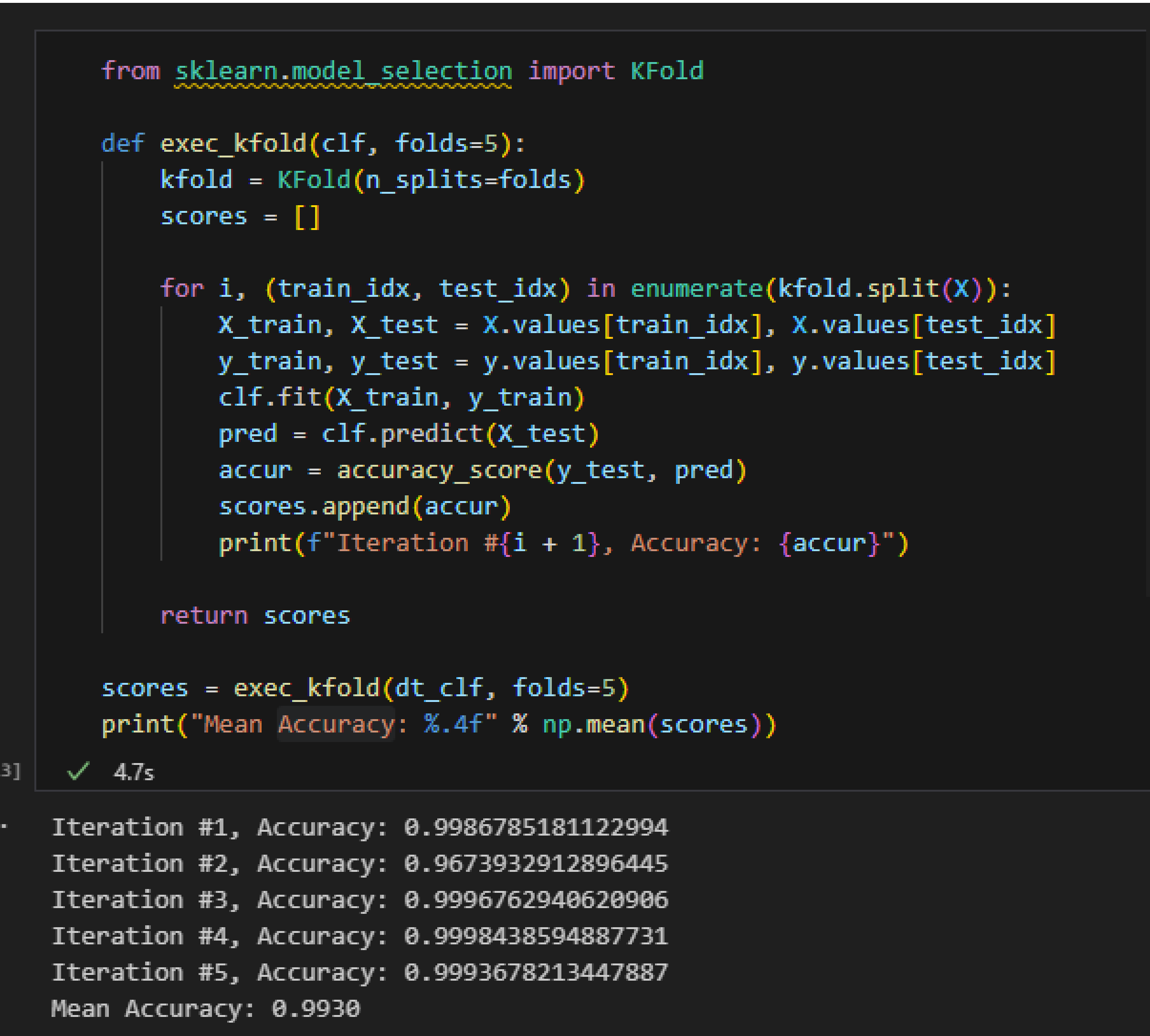
```
RandomForestClassifier Accuracy: 0.9930
```

```python
lr_clf.fit(X_train , y_train)
lr_pred = lr_clf.predict(X_test)
print("LogisticRegression Accuracy: %.4f" % accuracy_score(y_test, lr_pred))
```

[22] ✓ 4.3s Python

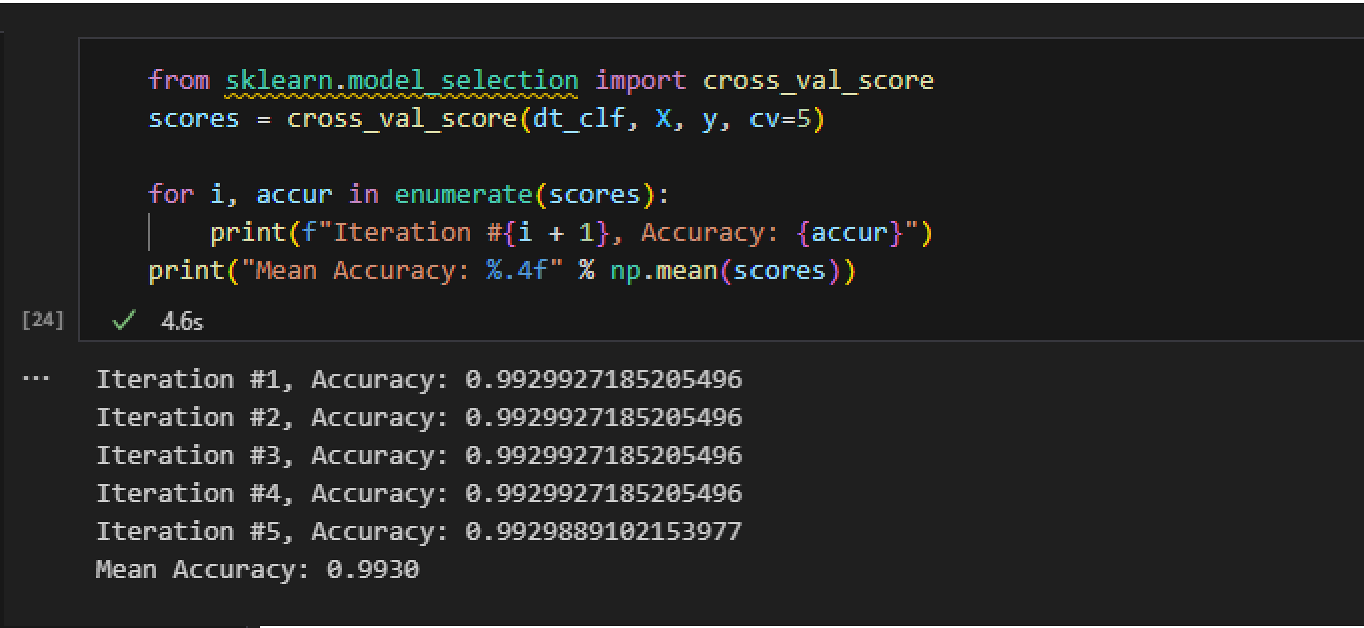
```
LogisticRegression Accuracy: 0.9930
```

분류기 피팅 및 정확도 평가



```python
from sklearn.model_selection import KFold

def exec_kfold(clf, folds=5):
    kfold = KFold(n_splits=folds)
    scores = []

    for i, (train_idx, test_idx) in enumerate(kfold.split(X)):
        X_train, X_test = X.values[train_idx], X.values[test_idx]
        y_train, y_test = y.values[train_idx], y.values[test_idx]
        clf.fit(X_train, y_train)
        pred = clf.predict(X_test)
        accur = accuracy_score(y_test, pred)
        scores.append(accur)
        print(f"Iteration #{i + 1}, Accuracy: {accur}")

    return scores

scores = exec_kfold(dt_clf, folds=5)
print("Mean Accuracy: %.4f" % np.mean(scores))
```

```
Iteration #1, Accuracy: 0.9986785181122994
Iteration #2, Accuracy: 0.9673932912896445
Iteration #3, Accuracy: 0.9996762940620906
Iteration #4, Accuracy: 0.9998438594887731
Iteration #5, Accuracy: 0.9993678213447887
Mean Accuracy: 0.9930
```



```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(dt_clf, X, y, cv=5)

for i, accur in enumerate(scores):
    print(f"Iteration #{i + 1}, Accuracy: {accur}")
print("Mean Accuracy: %.4f" % np.mean(scores))
```

```
Iteration #1, Accuracy: 0.9929927185205496
Iteration #2, Accuracy: 0.9929927185205496
Iteration #3, Accuracy: 0.9929927185205496
Iteration #4, Accuracy: 0.9929927185205496
Iteration #5, Accuracy: 0.9929889102153977
Mean Accuracy: 0.9930
```

실험 반성

- NaN값을 더 많이 생성하고 Mean 값으로 채운 행동
- 정확도를 낮추고 싶었으면 정답 레이블에는 수행하면 안되었음

- 틀린 데이터가 정답이 된 데이터를 학습하였을 뿐임