기상상태에 따른 울산 지역 닻 끌림 발생 여부 분석

이승윤 이창희 이지연 이민재 심세은

#1

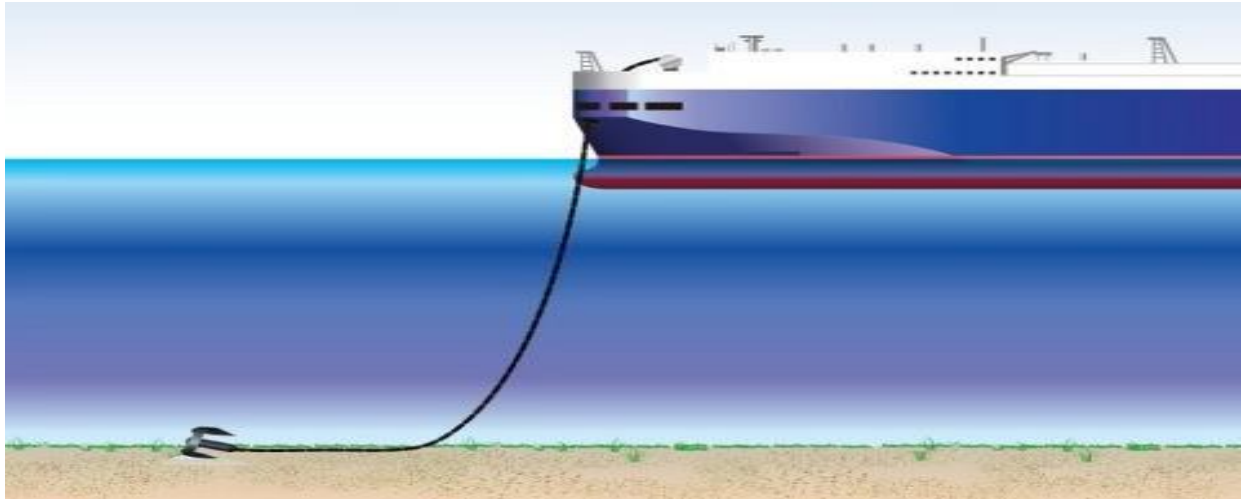
분석 주제

주제 선정 이유

- 대량의 Raw 데이터를 토대로 분석하고자 하는 의견이 있었음
- 상대적으로 데이터 접근이 용이한 공공 데이터를 활용하여 진행을 하고자 함
- 본 공모전이 학습한 내용을 바탕으로 진행하기에 알맞다고 판단함

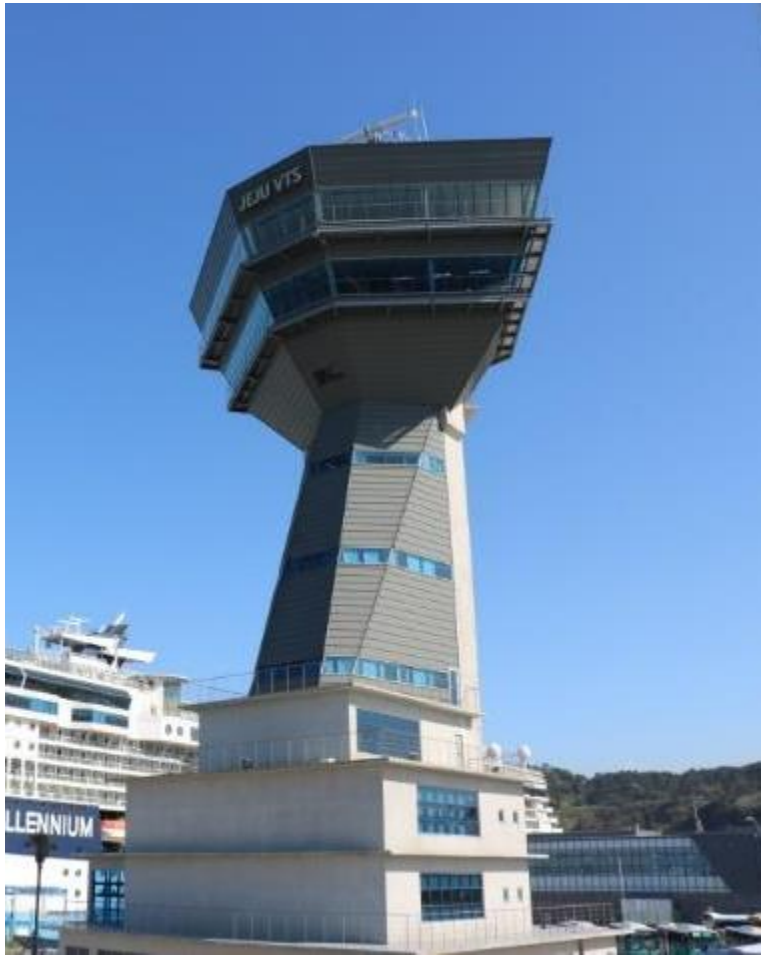
닻 끌림이란?

- 해류 · 조류, 바람 등 기상의 영향으로 해저의 닻이 끌리면서 선박의 위치가 고정되지 않고 이동하는 현상
- 주로 기상악화 시 동시다발적으로 발생하여 구조작업에 난항
- 해양사고의 가능성이 매우 높음



닻 끌림 인지 과정

- VTS가 정박지 관찰
- 선박의 위치가 정박지 내에 있는지,
  선박의 항적이 선회반경과 유사한지 감시



#2

데이터 설명

테이블명	설명
ulsan_anch_train_final (169,631)	정박 데이터
ulsan_drag_train_final (218,612)	닻 끌림 발생 데이터
ulsan_anch_drag_test (186,513)	테스트 데이터 (정박 + 닻끌림)
khnp_buoy_train	파고부이 측정 데이터 (한국수력원자력 측정 자료)
khnp_buoy_test	
kma_lightbecon_train	등표 측정 데이터
kma_lightbecon_test	
kma_pagobuoy_train	파고부이 측정 데이터 (기상청 측정 자료)
kma_pagobuoy_test	

선박 데이터

테이블명	속성	설명
ulsan_anch_train_final (정박 데이터) ulsan_drag_train_final (닻끌림 발생 데이터) ulsan_anch_drag_test (정박 + 닻끌림 데이터)	num	선박번호
	time	데이터 발생 시간
	latitude	선박이 위치한 위도 좌표
	longitude	선박이 위치한 경도 좌표
	sog	대지속력
	cog	실침로
	hdg	선수미선

날씨 데이터

테이블명	속성	설명
khnp_buoy_train khnp_buoy_test kma_lightbecon_train kma_lightbecon_test kma_pagobuoy_train kma_pagobuoy_test	yyyymmddhhmi	시간 (연/월/일/시/분/초)
	stn	데이터 관측 지점 번호
	stn_name	데이터 관측 지점명
	ws	유향/풍향 (deg)
	wd	유속/풍속 (m/s)
	max_wh	최대파고 (m)
	sig_wh	유의파고 (m)
	mean_wh	파고의 평균값 (m)

TEST 데이터

테이블명	속성	설명
ulsan_anch_drag_test (정박 + 닻끌림 데이터)	num	선박번호
	area	닻 끌림이 발생한 장소
	time	데이터 발생 시간
	latitude	선박이 위치한 위도 좌표
	longitude	선박이 위치한 경도 좌표
	sog	대지속력
	cog	실침로

#3

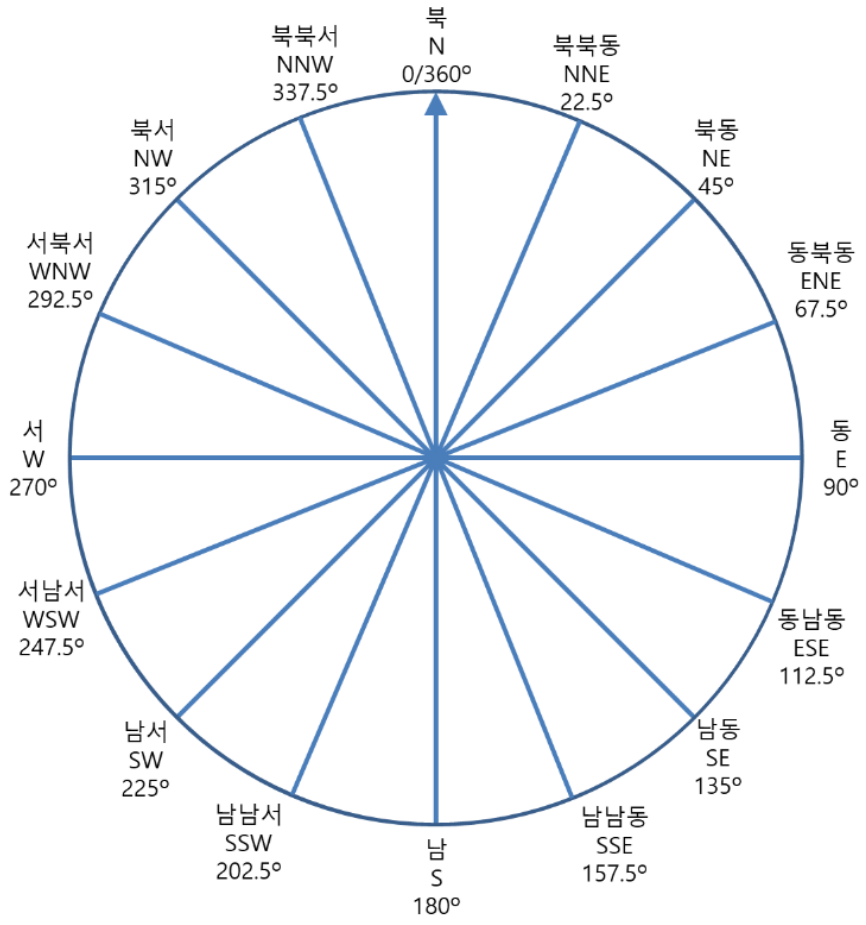
데이터 전처리

시간 데이터의 표기 통일

ulsan_anch_drag_train.time	kma_pagobuoy_train.yyyymmddhhmi	khnp_buoy_train.yyyymmddhhmi
"2022-08-14 00:55:06"	2021010100	202101010000
"2022-08-14 00:55:16"	2021010101	202101010001
연/월/일/시/분/초	연/월/일/시	연/월/일/시/분

year	month	day	hour	min	sec
2021	1	3	11	8	37
2021	1	3	11	11	36

결측치 처리



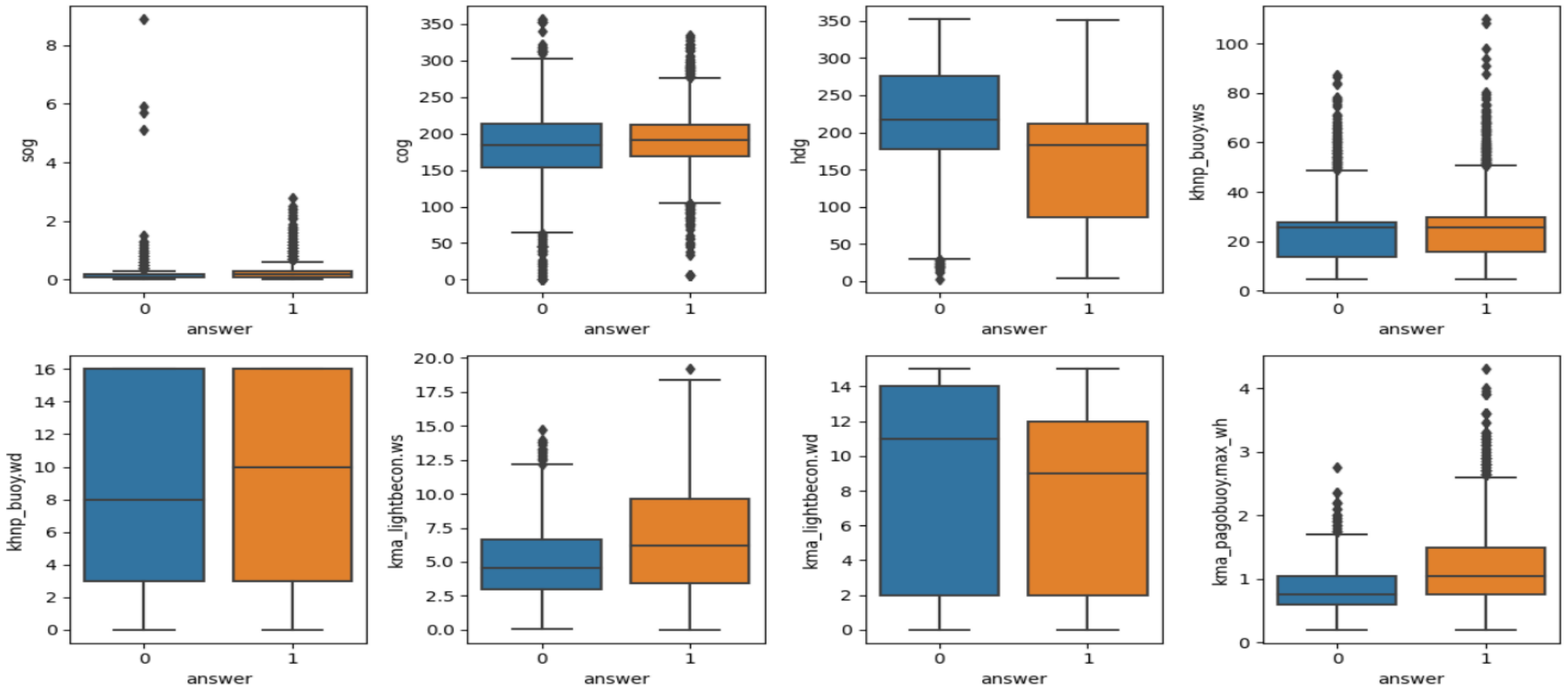
- **유향/풍향 결측치**

  Data Set 별로 방위 표시(16방위/360방위)가 달라
  360방위를 16방위로 변경 후
  시간대 별로 최빈값을 구하여 결측치 대체

- **유속/풍속 결측치**

  시간대별로 평균값을 구하여 결측치 대체

이상치 제거


sog
cog
hdg
khnp_buoy.ws
khnp_buoy.wd
kma_lightbecon.ws
kma_lightbecon.wd
kma_pagobuoy.max_wh
answer
0
1

데이터 통합

ulsan_anch/drag_train (특성 12개)

num	year	...	sog	cog	hdg
1	2021	...	0.6	102.9	343
1	2021	...	0.9	70.6	299
1	2021	...	0.4	9.6	273
1	2021	...	0.2	328	267
1	2021	...	0.2	260.1	265
1	2021	...	0.1	262.8	270

khnp_buoy_train (특성 8개)

stn_name	year	...	wd	ws
Gori	2021	...	84	37
Gori	2021	...	88	27.8
Gori	2021	...	84	39.3
Gori	2021	...	86	26.5
Gori	2021	...	85	36.6
Gori	2021	...	79	30.6

kma_lightbecon_train (특성 8개)

stn_name	year	...	wd	ws
Idukseo	2021	...	191	3.4
Idukseo	2021	...	127	5.2
Idukseo	2021	...	99	5.4
Idukseo	2021	...	110	5.3
Idukseo	2021	...	51	5.3
Idukseo	2021	...	135	4.5

kma_pagobuoy_train (특성 8개)

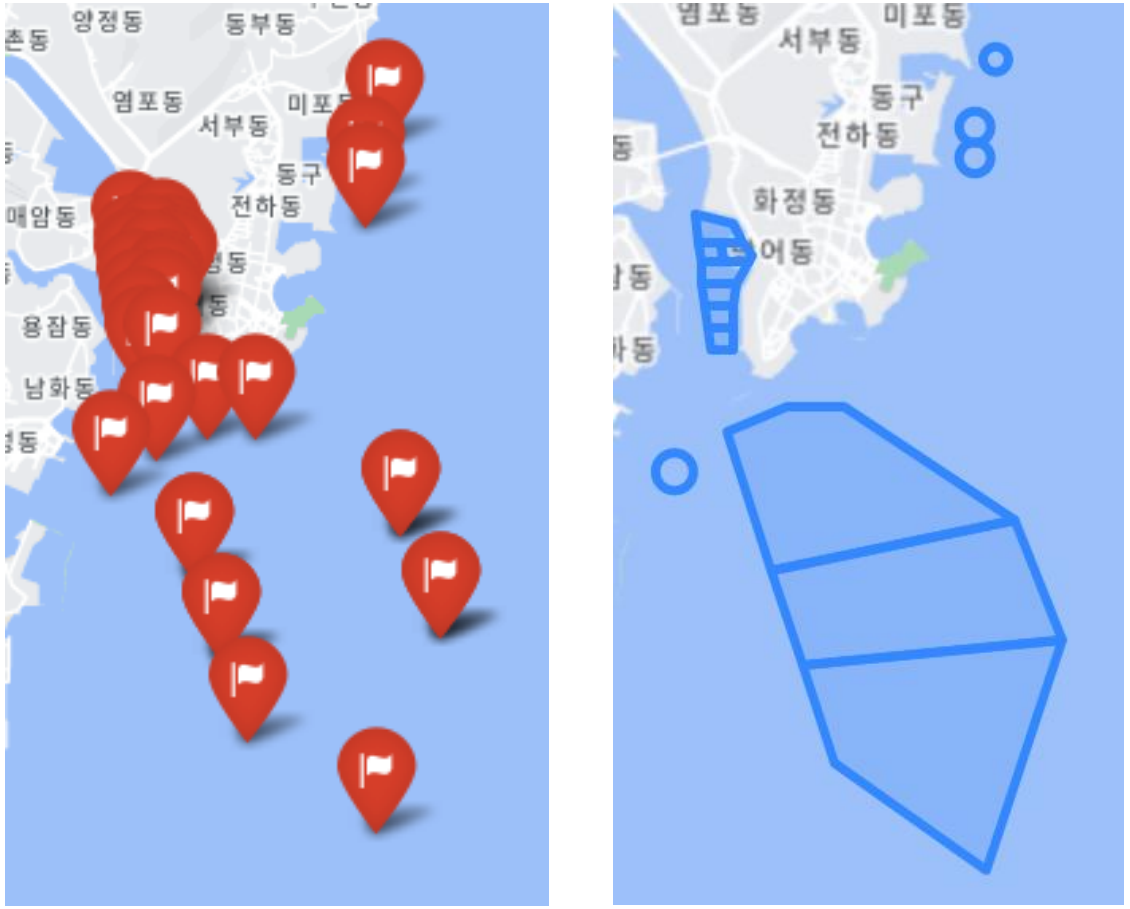
stn_name	year	...	max_wh	sig_wh	mean_wh
간절곶	2021	...	1	0.8	0.6
간절곶	2021	...	1	0.8	0.5
간절곶	2021	...	1.2	0.8	0.5
간절곶	2021	...	1	0.8	0.5
간절곶	2021	...	1.1	0.8	0.5
간절곶	2021	...	1	0.7	0.5

num	year	month	day	hour	min	sec	sog	cog	hdg	khnp_buoy.stn_name	khnp_buoy.ws	khnp_buoy.wd	kma_lightbecon.stn_name	kma_lightbecon.ws	kma_lightbecon.wd	kma_pagobuoy.max_wh	kma_pagobuoy.sig_wh	kma_pagobuoy.mean_wh	answer
1	2021	1	3	11	8	37	0.6	102.9	343	Gori	37	84	Idukseo	3.4	191	1	0.8	0.6	0
1	2021	1	3	11	11	36	0.9	70.6	299	Gori	27.8	88	Idukseo	5.2	127	1	0.8	0.5	0
1	2021	1	3	11	14	38	0.4	9.6	273	Gori	39.3	84	Idukseo	5.4	99	1.2	0.8	0.5	0
1	2021	1	3	11	17	33	0.2	328	267	Gori	26.5	86	Idukseo	5.3	110				0
1	2021	1	3	11	20	37	0.2	260.1	265	Gori	36.6	85	Idukseo	5.3	51	1.1	0.8	0.5	0
1	2021	1	3	11	23	38	0.1	262.8	270				Idukseo	4.5	135	1	0.7	0.5	0

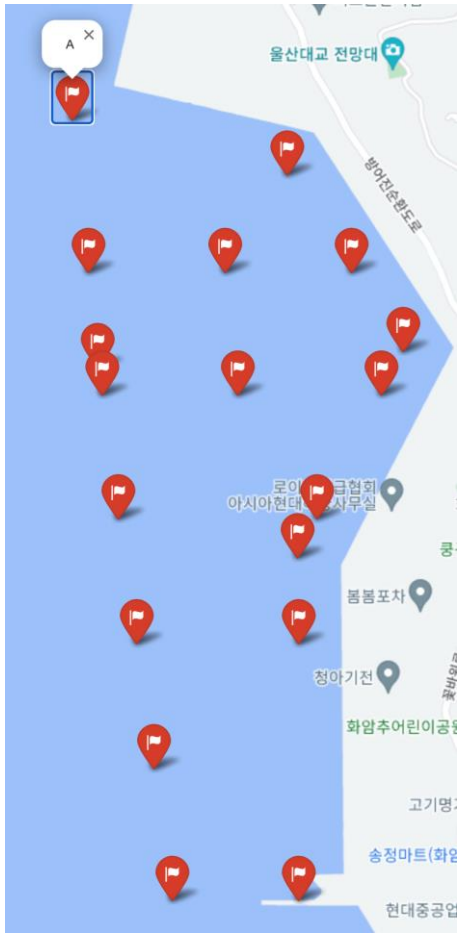
#4

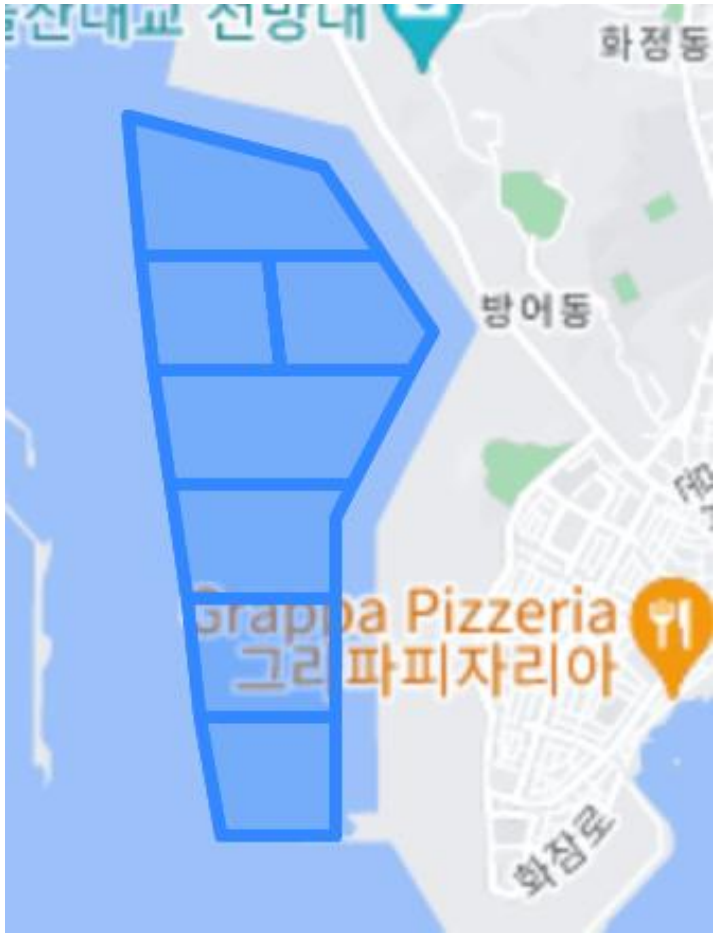
EDA

데이터 분포 시각화 - 닻 끌림 데이터



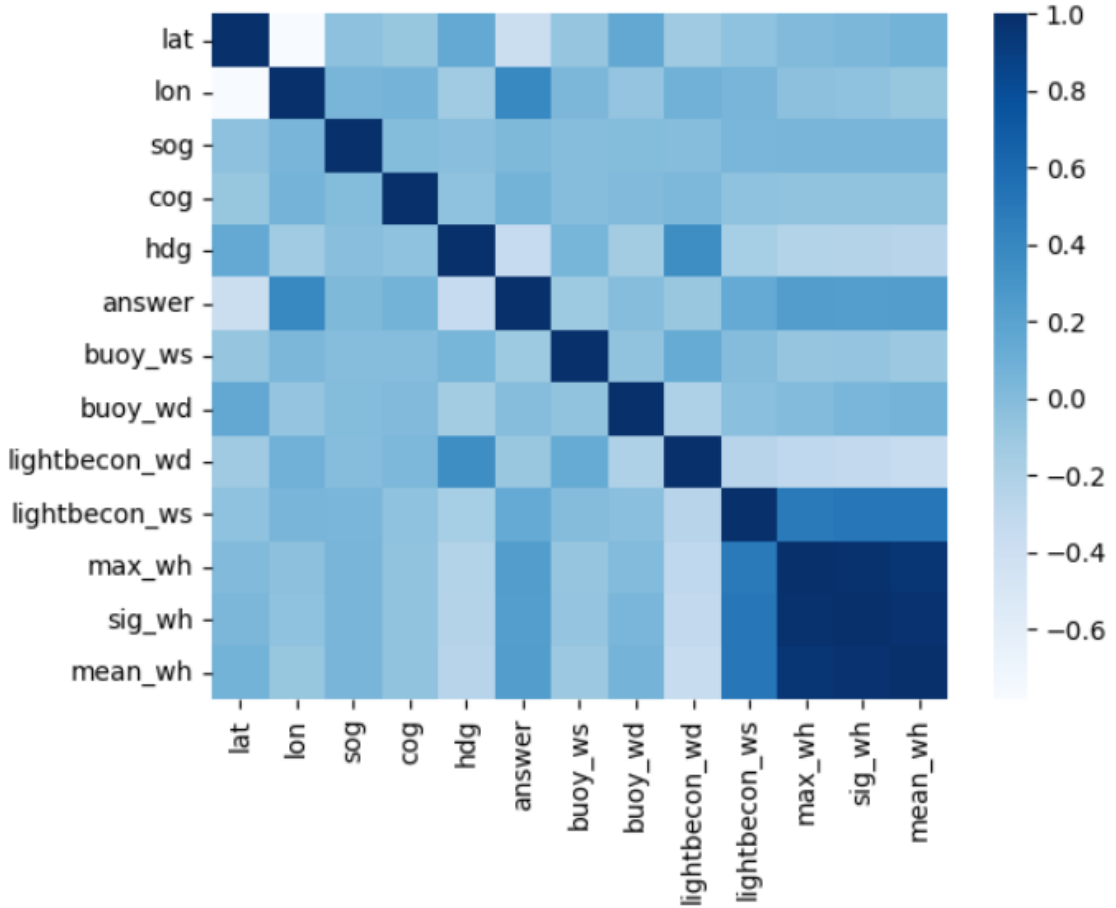
데이터 분포 시각화 - 닻 끌림 데이터





데이터 분포 시각화 - 닻 끌림 데이터

상관관계 분석


lat
lon
sog
cog
hdg
answer
buoy_ws
buoy_wd
lightbecon_wd
lightbecon_ws
max_wh
sig_wh
mean_wh
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

#5

분석 기법

알고리즘만 사용 (sec 단위)

- RandomForest Classifier

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v2.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'num'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier()

# 모델 훈련
rf.fit(X_train, y_train)
```

훈련 세트 정확도: 1.0
검증 세트 정확도: 0.9999

- XGBoost Classifer

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v2.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'num'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
xgb = XGBClassifier()

# 모델 훈련
xgb.fit(X_train, y_train)
```

훈련 세트 정확도: 1.0
검증 세트 정확도: 0.9999

max_depth = 10 (sec 단위)

- RandomForest Classifier

```
df_train = pd.read_csv('./4. 통합 데이터 전처리/
df_train_v2.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'num'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data,
target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier(max_depth=10)

# 모델 훈련
rf.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9996
검증 세트 정확도: 0.9997

- XGBoost Classifer

  → max_depth 설정할 수 없음

max_depth = 10 (min 단위)

- RandomForest Classifier

```
df_train = pd.read_csv('./4. 통합 데이터 전처리/
df_train_v3_1m.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data,
target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier(n_estimators=100,
max_depth=10, n_jobs=1)

# 모델 훈련
rf.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9984
검증 세트 정확도: 0.9979

- XGBoost Classifer

  → max_depth 설정할 수 없음

Standardscaler

- RandomForest Classifier

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v3_1m.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# 스케일링된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split(data_scaled, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier(n_estimators=100, max_depth=10, n_jobs=-1, random_state=42) # max_depth 13으로 바꿈

# 모델 훈련
rf.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9987
검증 세트 정확도: 0.9986

- XGBoost Classifer

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v3_1m.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# 스케일링된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split(data_scaled, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
xgb = XGBClassifier(n_estimators=100, max_depth=10, n_jobs=-1, random_state=42) # max_depth 13으로 바꿈

# 모델 훈련
xgb.fit(X_train, y_train)
```

훈련 세트 정확도: 1.0
검증 세트 정확도: 0.9999

Standardscaler + 시간컬럼제거

- RandomForest Classifier

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v3_1m.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'year', 'month', 'day', 'hour', 'min'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# 스케일링된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split(data_scaled, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier(n_estimators=100, max_depth=10, n_jobs=-1, random_state=42) # max_depth 13으로 바꿈

# 모델 훈련
rf.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9720
검증 세트 정확도: 0.9689

- XGBoost Classifer

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v3_1m.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'year', 'month', 'day', 'hour', 'min'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# 스케일링된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split(data_scaled, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
xgb = XGBClassifier(n_estimators=100, max_depth=10, n_jobs=-1, random_state=42) # max_depth 13으로 바꿈

# 모델 훈련
xgb.fit(X_train, y_train)
```

훈련 세트 정확도: 1.0
검증 세트 정확도: 0.9985

Standardscaler + 차원축소(PCA)

- RandomForest Classifier

```python
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# PCA를 사용하여 차원 축소
pca = PCA(n_components = 10)  # 목표 차원 수로 조정
data_pca = pca.fit_transform(data_scaled)

# 축소된 데이터로 새로운 데이터프레임 생성
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9',
'PC10'])

# 축소된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split
(df_pca, target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier(n_estimators=100,
max_depth=10, n_jobs=-1, random_state=42) # max_depth
13으로 바꿈
```

훈련 세트 정확도: 0.9317
검증 세트 정확도: 0.9260

- XGBoost Classifer

```python
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# PCA를 사용하여 차원 축소
pca = PCA(n_components = 10)  # 목표 차원 수로 조정
data_pca = pca.fit_transform(data_scaled)

# 축소된 데이터로 새로운 데이터프레임 생성
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9',
'PC10'])

# 축소된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split
(df_pca, target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
xgb = XGBClassifier(n_estimators=100, max_depth=10,
n_jobs=-1, random_state=42) # max_depth 13으로 바꿈
```

훈련 세트 정확도: 0.9999
검증 세트 정확도: 0.9867

Standardscaler + 시간컬럼제거 + 차원축소

- RandomForest Classifier

```
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'year',
'month', 'day', 'hour', 'min'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# PCA를 사용하여 차원 축소
pca = PCA(n_components = 10)  # 목표 차원 수로 조정
data_pca = pca.fit_transform(data_scaled)

# 축소된 데이터로 새로운 데이터프레임 생성
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9',
'PC10'])

# 축소된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split
(df_pca, target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
rf = RandomForestClassifier(n_estimators=100,
max_depth=10, n_jobs=-1, random_state=42) # max_depth
13으로 바꿈
```

훈련 세트 정확도: 0.9521
검증 세트 정확도: 0.9430

- XGBoost Classifer

```
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'year',
'month', 'day', 'hour', 'min'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# PCA를 사용하여 차원 축소
pca = PCA(n_components = 10)  # 목표 차원 수로 조정
data_pca = pca.fit_transform(data_scaled)

# 축소된 데이터로 새로운 데이터프레임 생성
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9',
'PC10'])

# 축소된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split
(df_pca, target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
xgb = XGBClassifier(n_estimators=100, max_depth=10,
n_jobs=-1, random_state=42) # max_depth 13으로 바꿈
```

훈련 세트 정확도: 1.0
검증 세트 정확도: 0.9922

XGBRF Classifier

- 알고리즘만 사용 (sec 단위)

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v2.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'num'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier()

# 모델 훈련
xgbrfc.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9963
검증 세트 정확도: 0.9964

XGBRF Classifier

- max_depth = 10 (sec 단위)

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v2.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'num'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier(max_depth=10)

# 모델 훈련
xgbrfc.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9999
검증 세트 정확도: 0.9999

- max_depth = 10 (min 단위)

```python
df_train = pd.read_csv('./4. 통합 데이터 전처리/df_train_v3_1m.csv')

# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 훈련데이터와 검증데이터로 나누기
X_train, X_val, y_train, y_val = train_test_split(data, target, test_size=0.2, random_state=123, stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier(n_estimators=100, max_depth=10, n_jobs=1)

# 모델 훈련
xgbrfc.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9989
검증 세트 정확도: 0.9989

XGBRF Classifier

- Standardscaler

```
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# 스케일링된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val =
train_test_split(data_scaled, target,
test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier(n_estimators=100,
max_depth=10, n_jobs=-1, random_state=42) #
max_depth 13으로 바꿈

# 모델 훈련
xgbrfc.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9983
검증 세트 정확도: 0.9980

- Standardscaler + 시간컬럼제거

```
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'year',
'month', 'day', 'hour', 'min'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# 스케일링된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val =
train_test_split(data_scaled, target,
test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier(n_estimators=100,
max_depth=10, n_jobs=-1, random_state=42) #
max_depth 13으로 바꿈

# 모델 훈련
xgbrfc.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9710
검증 세트 정확도: 0.9675

XGBRF Classifier

- Standardscaler + 차원축소(PCA)

```python
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# PCA를 사용하여 차원 축소
pca = PCA(n_components = 10)  # 목표 차원 수로 조정
data_pca = pca.fit_transform(data_scaled)

# 축소된 데이터로 새로운 데이터프레임 생성
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9',
'PC10'])

# 축소된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split
(df_pca, target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier(n_estimators=100,
max_depth=10, n_jobs=-1, random_state=42) # max_depth
13으로 바꿈

# 모델 훈련
xgbrfc.fit(X_train, y_train)
```

훈련 세트 정확도: 0.9380
검증 세트 정확도: 0.9327

- Standardscaler + 시간컬럼제거 + 차원축소

```python
# 입력 데이터, 타깃 데이터, 테스트 데이터 설정
data = df_train.drop(columns=['answer', 'year',
'month', 'day', 'hour', 'min'])
target = df_train['answer']

# 스케일링
scaler = StandardScaler() #<- 스케일링 방식 변경 가능
data_scaled = scaler.fit_transform(data)

# PCA를 사용하여 차원 축소
pca = PCA(n_components = 10)  # 목표 차원 수로 조정
data_pca = pca.fit_transform(data_scaled)

# 축소된 데이터로 새로운 데이터프레임 생성
df_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2',
'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9',
'PC10'])

# 축소된 데이터로 모델 훈련 데이터 나누기
X_train, X_val, y_train, y_val = train_test_split
(df_pca, target, test_size=0.2, random_state=123,
stratify=target)

# 모델 객체 생성
xgbrfc = XGBRFClassifier(n_estimators=100,
max_depth=10, n_jobs=-1, random_state=42) # max_depth
13으로 바꿈
```

훈련 세트 정확도: 0.9536
검증 세트 정확도: 0.9465

모델별 성능 비교

	RandomForest	XGBoost	XGB RFC
기본	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9999	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9999	훈련 세트 정확도: 0.9964 검증 세트 정확도: 0.9965
max_depth = 10	훈련 세트 정확도: 0.9996 검증 세트 정확도: 0.9997	X	훈련 세트 정확도: 0.9999 검증 세트 정확도: 0.9999
max_depth = 10, 1분 단위	훈련 세트 정확도: 0.9984 검증 세트 정확도: 0.9979	X	훈련 세트 정확도: 0.9989 검증 세트 정확도: 0.9989
StandardScaler	훈련 세트 정확도: 0.9987 검증 세트 정확도: 0.9986	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9999	훈련 세트 정확도: 0.9983 검증 세트 정확도: 0.9981
StandardScaler + 시간제거	훈련 세트 정확도: 0.9720 검증 세트 정확도: 0.9689	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9985	훈련 세트 정확도: 0.9710 검증 세트 정확도: 0.9675
StandardScaler + 차원축소	훈련 세트 정확도: 0.9317 검증 세트 정확도: 0.9260	훈련 세트 정확도: 0.9999 검증 세트 정확도: 0.9867	훈련 세트 정확도: 0.9381 검증 세트 정확도: 0.9327
StandardScaler + 시간제거 + 차원축소	훈련 세트 정확도: 0.9521 검증 세트 정확도: 0.9430	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9922	훈련 세트 정확도: 0.9536 검증 세트 정확도: 0.9465

Ensemble 사용

모델 (뒤/앞)	XGB Classifier	LightGBM Classifier	RandomForest Classifier	Extra Trees Classifier	AdaBoost Classifier	XGBRF Classifier
XGB Classifier	X	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.967	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9239	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9684	훈련 세트 정확도: 0.9450 검증 세트 정확도: 0.9267	훈련 세트 정확도: 0.9723 검증 세트 정확도: 0.9353
LightGBM Classifier	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9339	X	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9209	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9669	훈련 세트 정확도: 0.9450 검증 세트 정확도: 0.9195	훈련 세트 정확도: 0.9723 검증 세트 정확도: 0.9310
RandomForest Classifier	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9239	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9209	X	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9181	훈련 세트 정확도: 0.9396 검증 세트 정확도: 0.8678	훈련 세트 정확도: 0.9622 검증 세트 정확도: 0.8865
Extra Trees Classifier	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9339	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.967	훈련 세트 정확도: 1.0 검증 세트 정확도: 0.9181	X	훈련 세트 정확도: 0.9450 검증 세트 정확도: 0.9167	훈련 세트 정확도: 0.9723 검증 세트 정확도: 0.9282
AdaBoost Classifier	훈련 세트 정확도: 0.9397 검증 세트 정확도: 0.8807	훈련 세트 정확도: 0.9450 검증 세트 정확도: 0.9195	훈련 세트 정확도: 0.9397 검증 세트 정확도: 0.8678	훈련 세트 정확도: 0.9397 검증 세트 정확도: 0.8807	X	훈련 세트 정확도: 0.8933 검증 세트 정확도: 0.8707
XGBRF Classifier	훈련 세트 정확도: 0.9601 검증 세트 정확도: 0.9023	훈련 세트 정확도: 0.9723 검증 세트 정확도: 0.9310	훈련 세트 정확도: 0.9622 검증 세트 정확도: 0.8864	훈련 세트 정확도: 0.9723 검증 세트 정확도: 0.9281	훈련 세트 정확도: 0.8624 검증 세트 정확도: 0.8348	X

하이퍼파라미터 튜닝 (Hyperparameter Tunning)

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

params = {'n_estimators': randint(50, 300),
    'max_depth': randint(3, 10),
    'learning_rate': uniform(0.01, 0.3),
    'subsample': uniform(0.5, 0.5),
    'colsample_bynode': uniform(0.5, 0.5),
    'reg_alpha': uniform(0, 1),
    'reg_lambda': uniform(0, 1),
    'min_child_weight': randint(1, 10),
    'gamma': uniform(0, 1),
    'scale_pos_weight': uniform(0, 1)}

gs = RandomizedSearchCV(XGBRFClassifier(random_state=42), params, n_iter=100, n_jobs=-1, random_state=42)
gs.fit(data_pca, target)


print(gs.best_params_)
```

```
{'colsample_bynode': 0.6650497566550777, 'gamma': 0.321582764680029, 'learning_rate': 0.03768717586862382, 'max_depth': 9, 'min_child_weigh
t': 9, 'n_estimators': 263, 'reg_alpha': 0.08175903194887191, 'reg_lambda': 0.8735786241067772, 'scale_pos_weight': 0.9208724005318132, 'su
bsample': 0.5305389799274318}
```

하이퍼파라미터 튜닝 (Hyperparameter Tunning)

```
# 모델 객체 생성
xgbrfc = XGBRFClassifier(colsample_bynode=0.6650497566550777, gamma= 0.321582764680029, learning_rate= 0.03768717586862382, max_depth= 9, min_child_weight= 9,
n_estimators= 263, reg_alpha= 0.08175903194887191, reg_lambda= 0.8735786241067772, scale_pos_weight= 0.9208724005318132, subsample= 0.5305389799274318)
# 모델 훈련
xgbrfc.fit(X_train, y_train)

# 훈련 세트 정확도 출력
train_pred = xgbrfc.predict(X_train)
train_accuracy = accuracy_score(y_train, train_pred)
print("훈련 세트 정확도:", train_accuracy)

# 검증 세트 정확도 출력
val_pred = xgbrfc.predict(X_val)
val_accuracy = accuracy_score(y_val, val_pred)
print("검증 세트 정확도:", val_accuracy)
```

✓ 15.4s

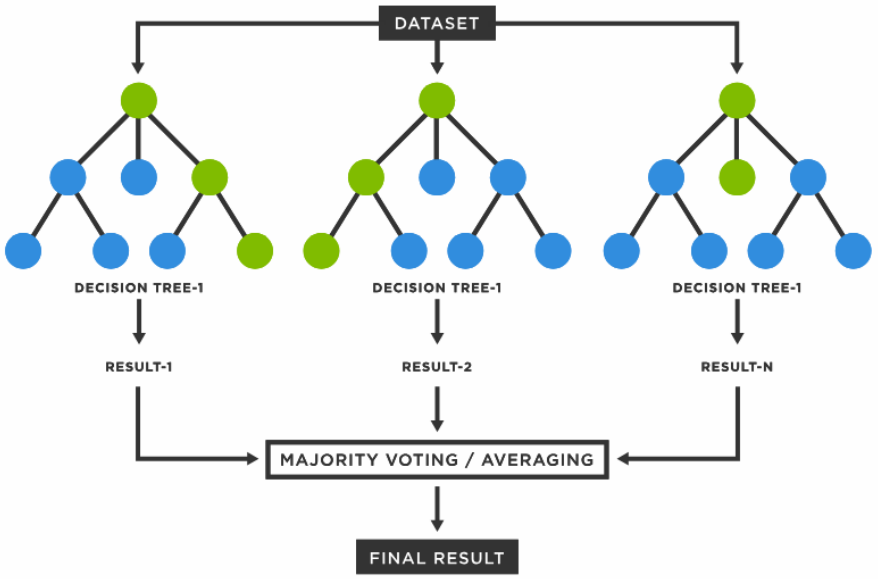
훈련 세트 정확도: 0.9380　　검증 세트 정확도: 0.9327

#6

분석 결과

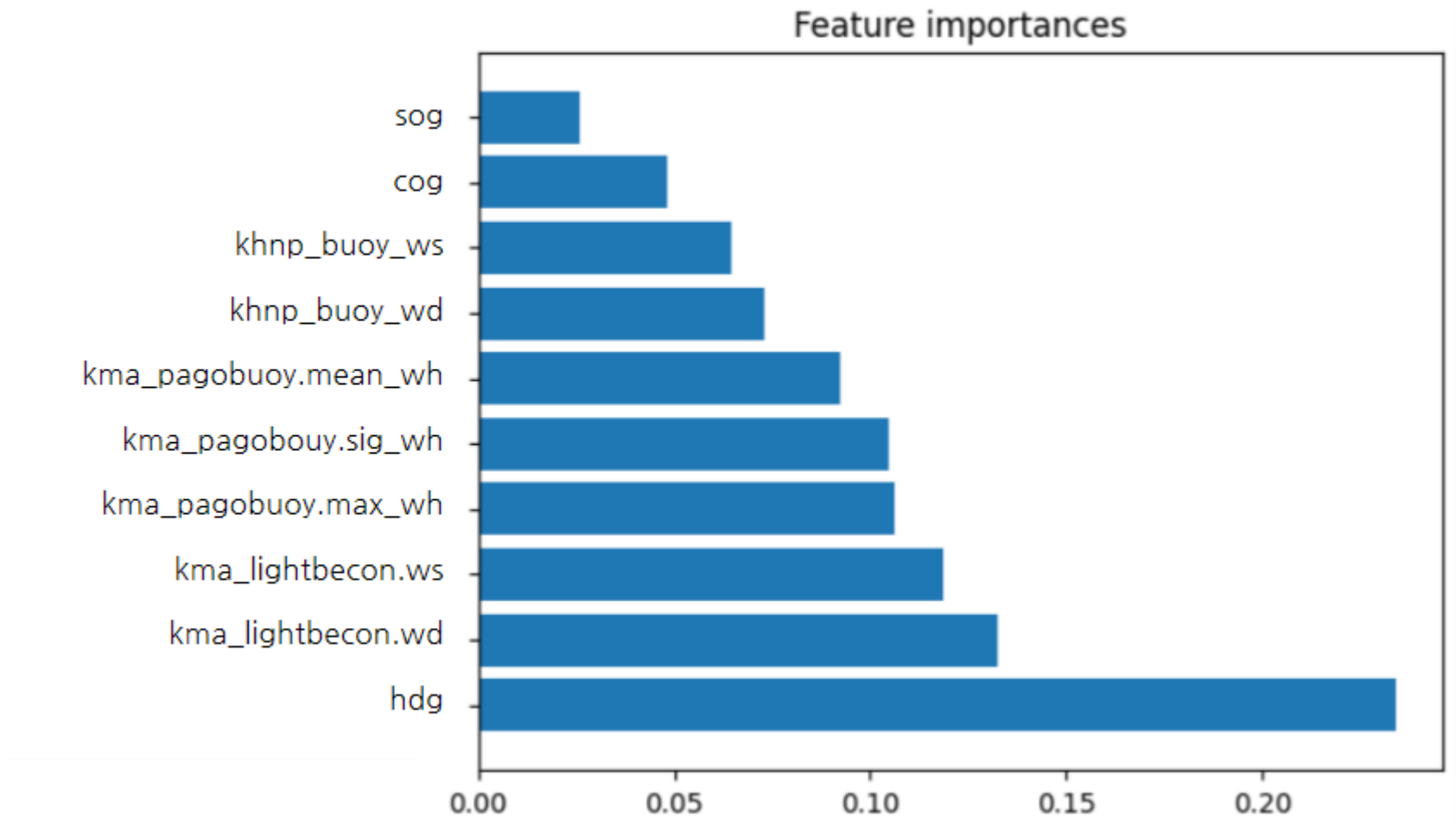
최종 선정 모델

XGBRF Classifier



parameter	parameter description	value
n_estimators	트리 개수 지정	263
max_depth	각 트리의 최대 깊이 지정	9
learning_rate	각 트리의 가중치 업데이트에 대한 학습 속도 지정	0.038
subsample	각 트리를 훈련할 때 사용할 샘플의 비율 지정	0.5305
colsample_bynode	각 분할에서 사용할 피처의 비율 지정	0.6650
reg_alpha	L1 정규화 항에 대한 가중치 지정	0.081
reg_lambda	L2 정규화 항에 대한 가중치 지정	0.8736
min_child_weight	리프 노드를 분할하기 위한 최소 가중치 합 지정	9
gamma	리프 노드의 손실 감소에 필요한 최소 손실 감소 값 지정	0.3216
scale_pos_weight	양성 클래스의 가중치 지정	0.9209

Feature Importances



감사합니다