



대구 교통사고 피해 예측 모델

입문초급팀: 무사고 기원
(강민정, 신유진, 최은빈)

목차

#01 EDA

#02 전처리

#03 모델링

#04 결론



1. EDA



#1-1. 데이터셋 탐색

#데이터 불러오기

```
train = pd.read_csv("./train.csv")
train.head()
```



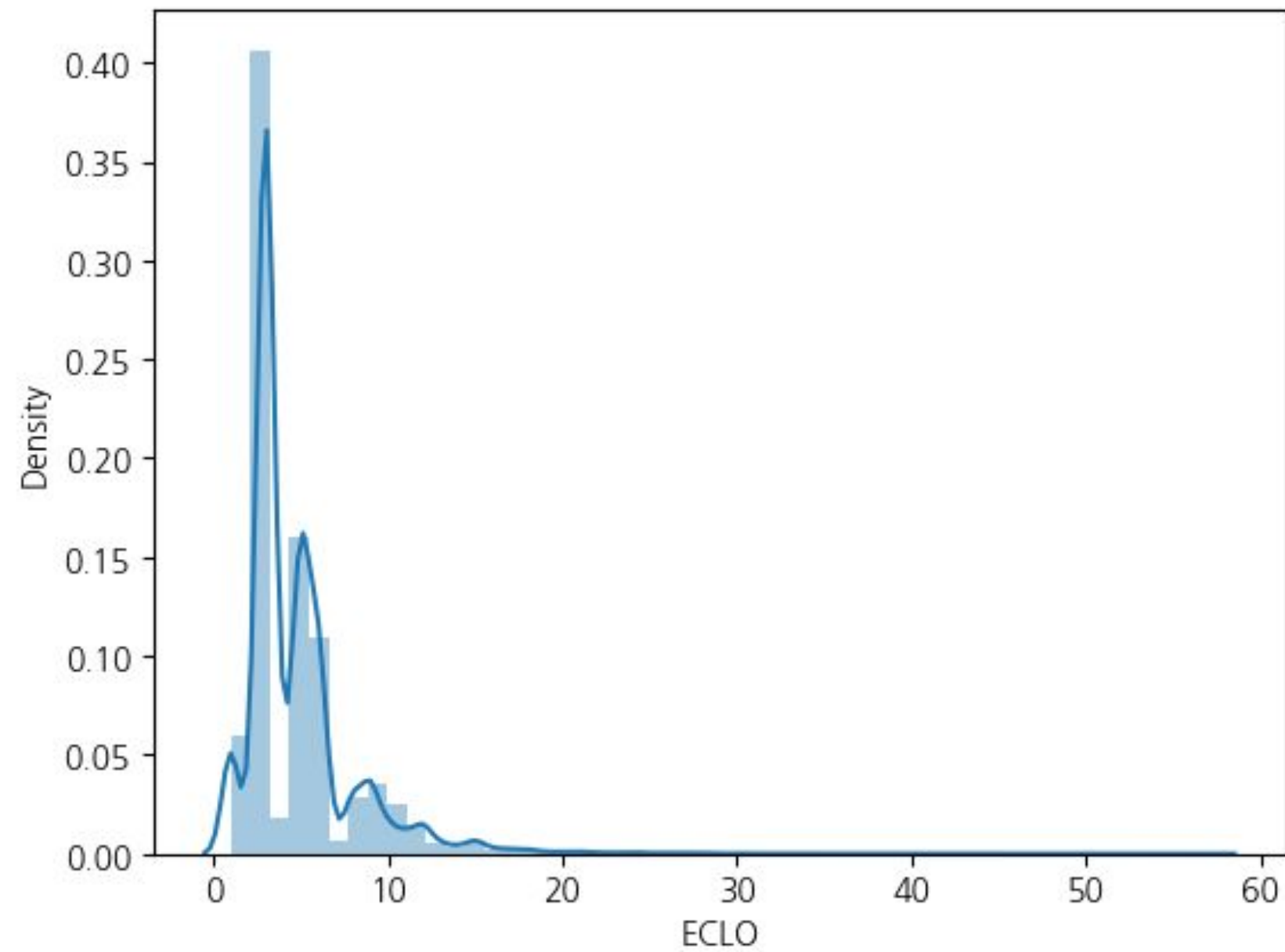
	ID	사고일시	요일	기상상태	시군구	도로형태	노면상태	사고유형	사고유형 - 세부분류	법규위반	...	가해운전자 상해정도	피해운전자 차종	피해운전자 성별	피해운전자 연령	피해운전자 상해정도	사망 자수	중상 자수	경상 자수	부상 자수	ECL0
0	ACCIDENT_00000	2019-01-01 00	화요일	맑음	대구광역시 중구 대신동	단일로 - 기타	건조	차대사람	길가장자리구역통행중	안전운전불이행	...	상해없음	보행자	여	70세	중상	0.0	1.0	0.0	0.0	5.0
1	ACCIDENT_00001	2019-01-01 00	화요일	흐림	대구광역시 달서구 감삼동	단일로 - 기타	건조	차대사람	보도통행중	기타	...	상해없음	보행자	남	61세	경상	0.0	0.0	1.0	0.0	3.0
2	ACCIDENT_00002	2019-01-01 01	화요일	맑음	대구광역시 수성구 두산동	단일로 - 기타	건조	차대사람	차도통행중	안전운전불이행	...	상해없음	보행자	남	38세	경상	0.0	0.0	1.0	0.0	3.0
3	ACCIDENT_00003	2019-01-01 02	화요일	맑음	대구광역시 북구 복현동	단일로 - 기타	건조	차대차	추돌	안전운전불이행	...	상해없음	승용	남	36세	중상	0.0	1.0	0.0	0.0	5.0
4	ACCIDENT_00004	2019-01-01 04	화요일	맑음	대구광역시 동구 신암동	단일로 - 기타	건조	차대차	추돌	안전운전불이행	...	상해없음	승용	남	52세	경상	0.0	0.0	1.0	0.0	3.0

train.columns

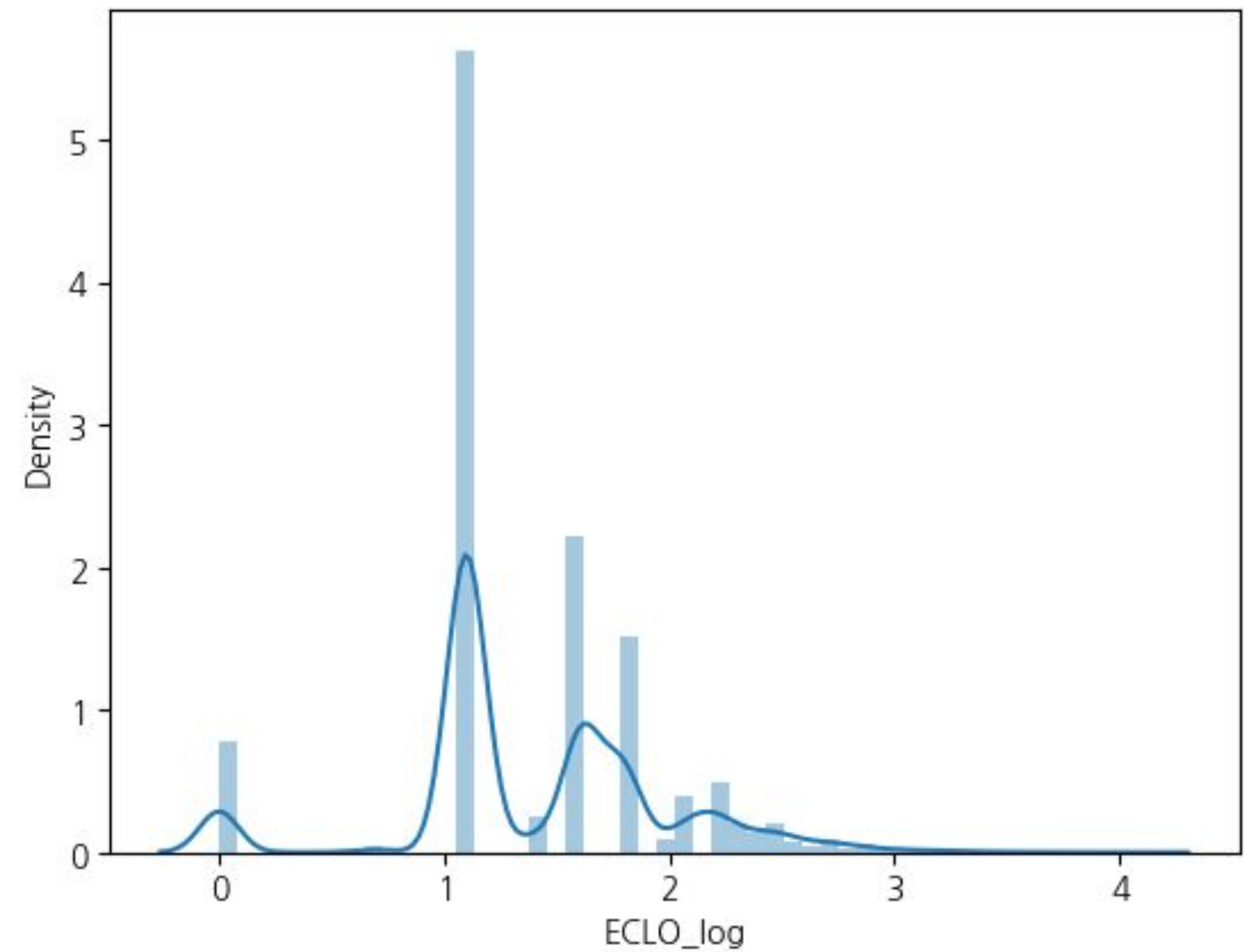
```
Index(['ID', '사고일시', '요일', '기상상태', '시군구', '도로형태', '노면상태', '사고유형',
      '사고유형 - 세부분류', '법규위반', '가해운전자 차종', '가해운전자 성별', '가해운전자 연령', '가해운전자 상해정도',
      '피해운전자 차종', '피해운전자 성별', '피해운전자 연령', '피해운전자 상해정도', '사망자수', '중상자수',
      '경상자수', '부상자수', 'ECL0'],
      dtype='object')
```

#1-2. EDA(target)

- ECLO



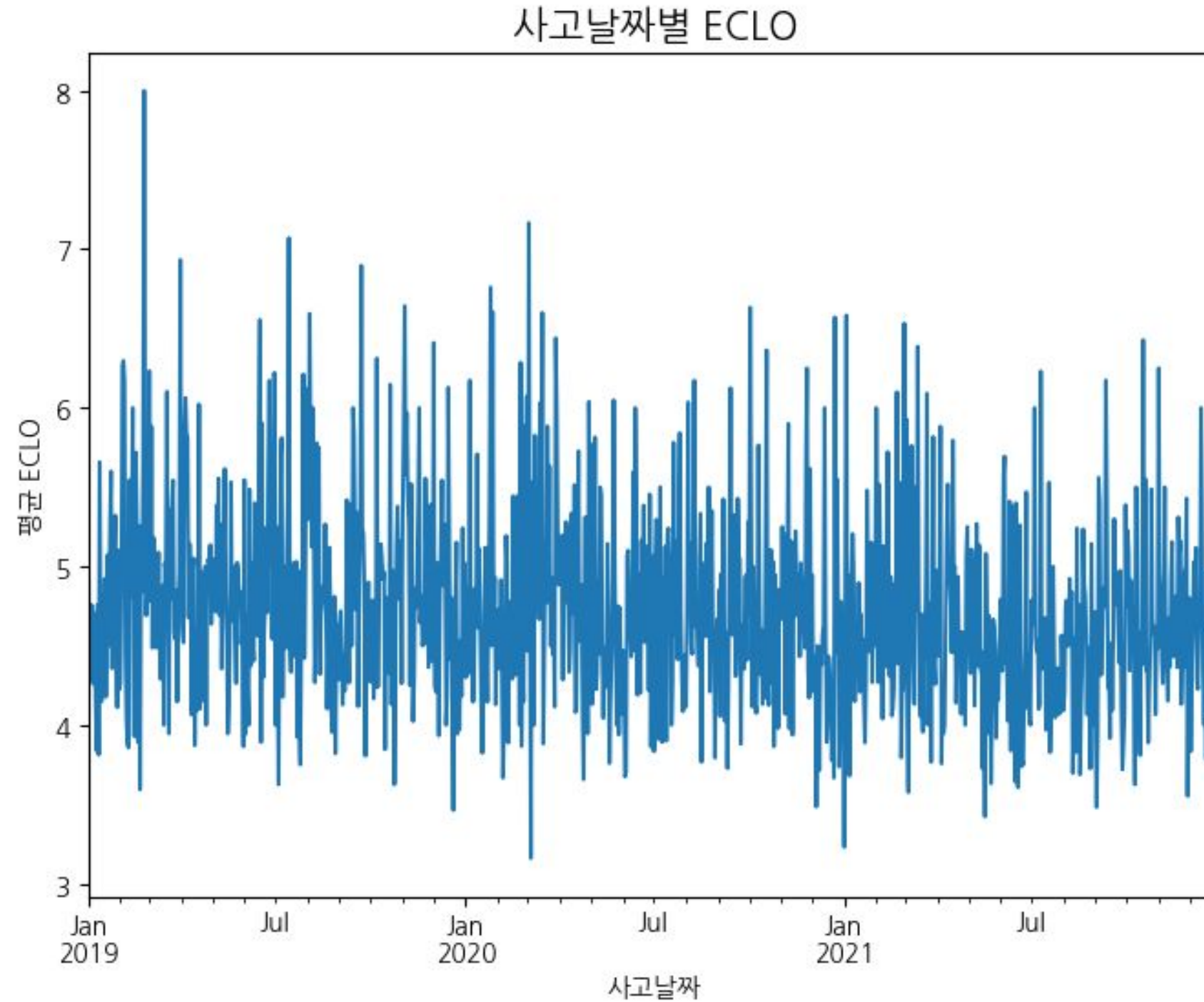
로그변환 전



로그변환 후

#1-3. EDA(사고일시)

1) 사고날짜



#1-3. EDA(사고일시)

1) 사고날짜 - '계절' Column 추가

```
#1-1.  
#어떤 계절에 사고가 많이 일어나는지 알아보기 위해 '계절' 열을 추가  
train['계절'] = (train['사고일시'].dt.month % 12 + 3) // 3  
train['계절'] = train['계절'].map({1: '겨울', 2: '봄', 3: '여름', 4: '가을'})  
  
#결과 확인  
train.head()
```

	ID	사고일시	요일	기상 상태	시군구	도로 형태	노면 상태	사고 유형	사고유형 - 세부 분류	법규위 반	...	피해운 전자 연령	피해운 전 자 상 해 정 도	사망 자수	중상 자수	경상 자수	부상 자수	ECL0	사고날 짜	사고 시간	계 절
0	ACCIDENT_00000	2019-01-01 00:00:00	화 요 일	맑음	대구광역시 중구 대신동	단일로 - 기타	건조	차대 사람	길가장자 리구역통 행중	안전운 전불이 행	...	70세	중상	0	1	0	0	5	2019- 01-01	00	겨 울
1	ACCIDENT_00001	2019-01-01 00:00:00	화 요 일	흐림	대구광역시 달서구 감동 삼동	단일로 - 기타	건조	차대 사람	보도통행 중	기타	...	61세	경상	0	0	1	0	3	2019- 01-01	00	겨 울
2	ACCIDENT_00002	2019-01-01 01:00:00	화 요 일	맑음	대구광역시 수성구 두 산동	단일로 - 기타	건조	차대 사람	차도통행 중	안전운 전불이 행	...	38세	경상	0	0	1	0	3	2019- 01-01	01	겨 울
3	ACCIDENT_00003	2019-01-01 02:00:00	화 요 일	맑음	대구광역시 북구 북현 동	단일로 - 기타	건조	차대 차	추돌	안전운 전불이 행	...	36세	중상	0	1	0	0	5	2019- 01-01	02	겨 울
4	ACCIDENT_00004	2019-01-01 04:00:00	화 요 일	맑음	대구광역시 동구 신암 동	단일로 - 기타	건조	차대 차	추돌	안전운 전불이 행	...	52세	경상	0	0	1	0	3	2019- 01-01	04	겨 울

#1-3. EDA(사고일시)

2) 공휴일 정보 추가

```
#1-2.
# '공휴일'에 사고가 많이 발생하는지 아닌지를 판단하기 위해 2018-2022년까지의 공휴일을 담은 외부 데이터셋 불러오기
holiday = pd.read_csv("./holiday18.csv", encoding = "cp949")
```

```
# 'train'과 'holiday'를 병합하고 '공휴일' 컬럼을 생성
train_1 = pd.merge(train, holiday[['날짜']], how='left', left_on='사고날짜', right_on='날짜')
train_1['공휴일'] = train_1['날짜'].notnull()

# '날짜' column 없애기
train = train_1.drop(columns = '날짜')
train.head()
```

	ID	요일	기상 상태	시군구	도로형 태	노면 상태	사고 유형	사고유형 - 세부분류	법규위 반	가해운 전자 차 종	...	피해운전 자 상해정 도	사망 자수	중상 자수	경상 자수	부상 자수	ECL0	사고날 짜	사고 시간	계 절	공 휴 일
0	ACCIDENT_00000	화 요 일	맑음	대구광역시 중구 대신동	단일로 - 기타	건조	차대 사람	길가장자리 구역통행중	안전운 전불이 행	승용	...	중상	0	1	0	0	5	2019- 01-01	00	겨 울	True
1	ACCIDENT_00001	화 요 일	흐림	대구광역시 달서구 감삼 동	단일로 - 기타	건조	차대 사람	보도통행중	기타	승용	...	경상	0	0	1	0	3	2019- 01-01	00	겨 울	True
2	ACCIDENT_00002	화 요 일	맑음	대구광역시 수성구 두산 동	단일로 - 기타	건조	차대 사람	차도통행중	안전운 전불이 행	승용	...	경상	0	0	1	0	3	2019- 01-01	01	겨 울	True
3	ACCIDENT_00003	화 요 일	맑음	대구광역시 북구 복현동	단일로 - 기타	건조	차대 차	추돌	안전운 전불이 행	승용	...	중상	0	1	0	0	5	2019- 01-01	02	겨 울	True
4	ACCIDENT_00004	화 요 일	맑음	대구광역시 동구 신암동	단일로 - 기타	건조	차대 차	추돌	안전운 전불이 행	승용	...	경상	0	0	1	0	3	2019- 01-01	04	겨 울	True

#1-3. EDA(사고일시)

2) 공휴일 정보 추가

cf) 공휴일에 ECLO가 높아질까?

```
[ ] holiday_eclo = train.groupby('공휴일')['ECLO'].mean().round(4)
    holiday_eclo
```

#공휴일에 평균 ECLO가 약 0.4만큼 높음.

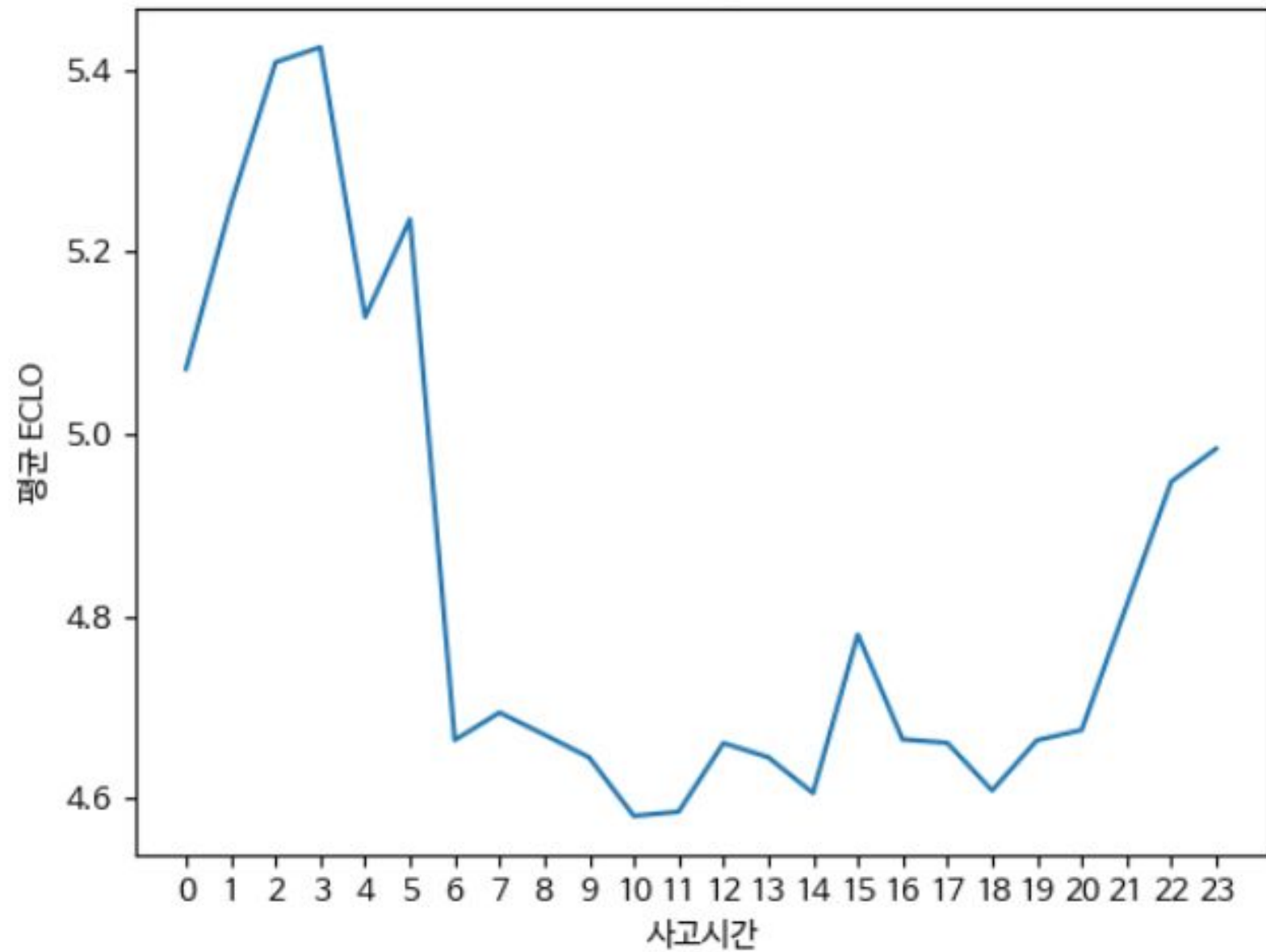
공휴일	ECLO
False	4.7118
True	5.1141

Name: ECLO, dtype: float64

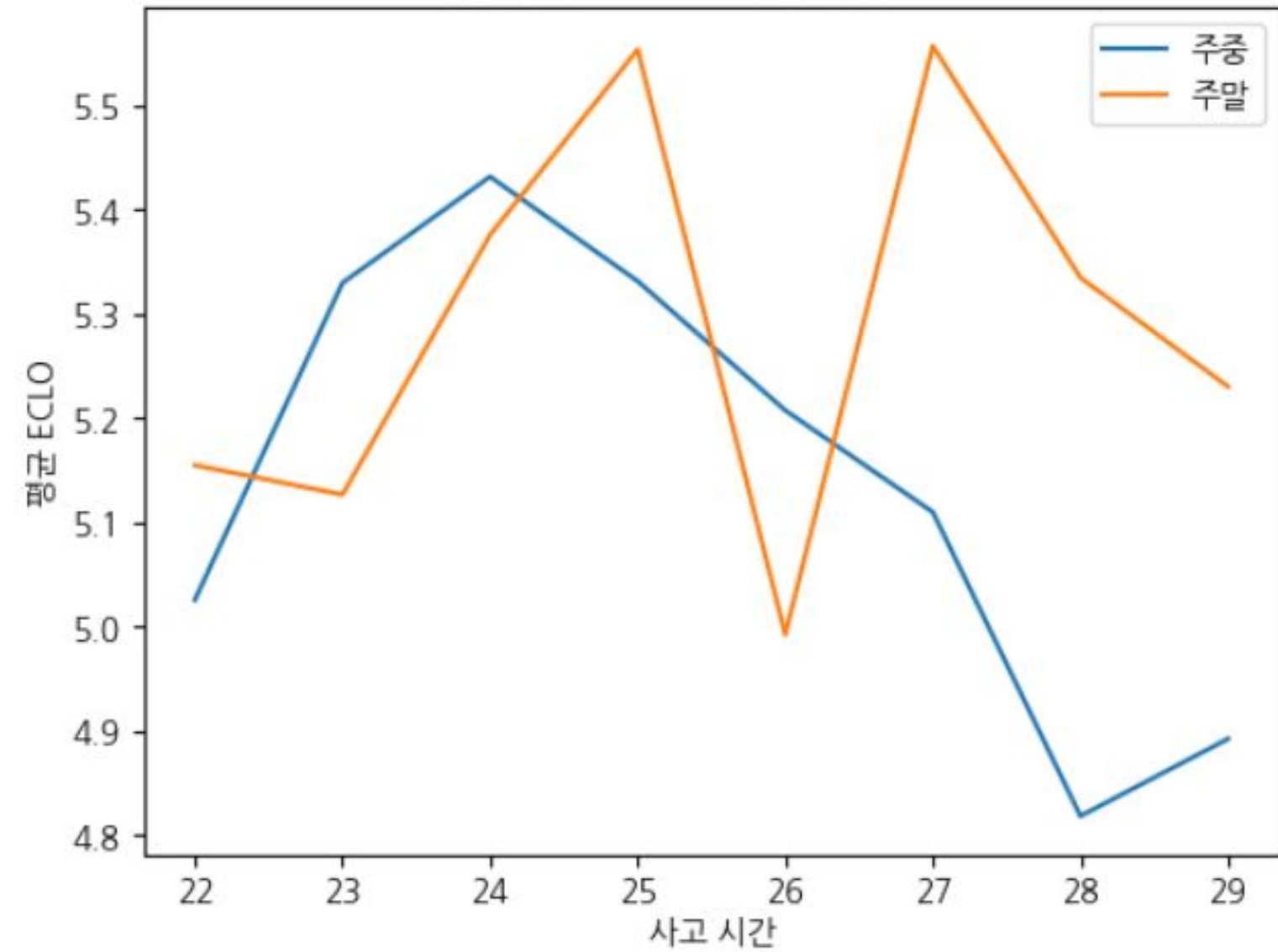
#1-3. EDA(사고일시)

3) 사고 시간

사고시간 별 평균 ECLO



주중 심야 ECLO vs 주말 심야 ECLO



#1-4. EDA(지역)

1) '동' 별 평균 ECLO

```
#지역 정보 추출
```

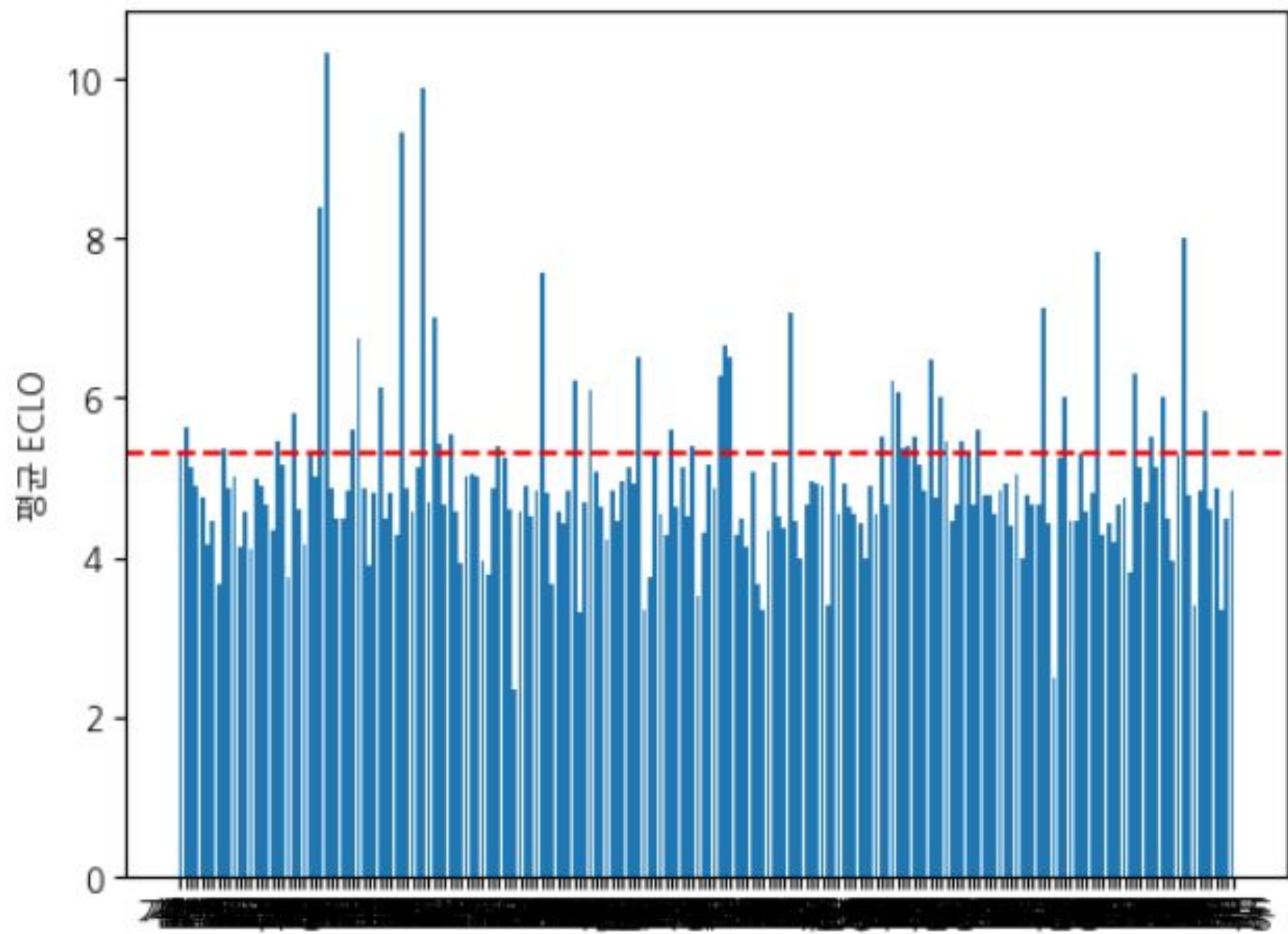
```
location_pattern = r'(\S+) (\S+) (\S+)'
```

```
train[['도시', '구', '동']] = train['시군구'].str.extract(location_pattern)
```

```
train = train.drop(columns=['시군구'])
```

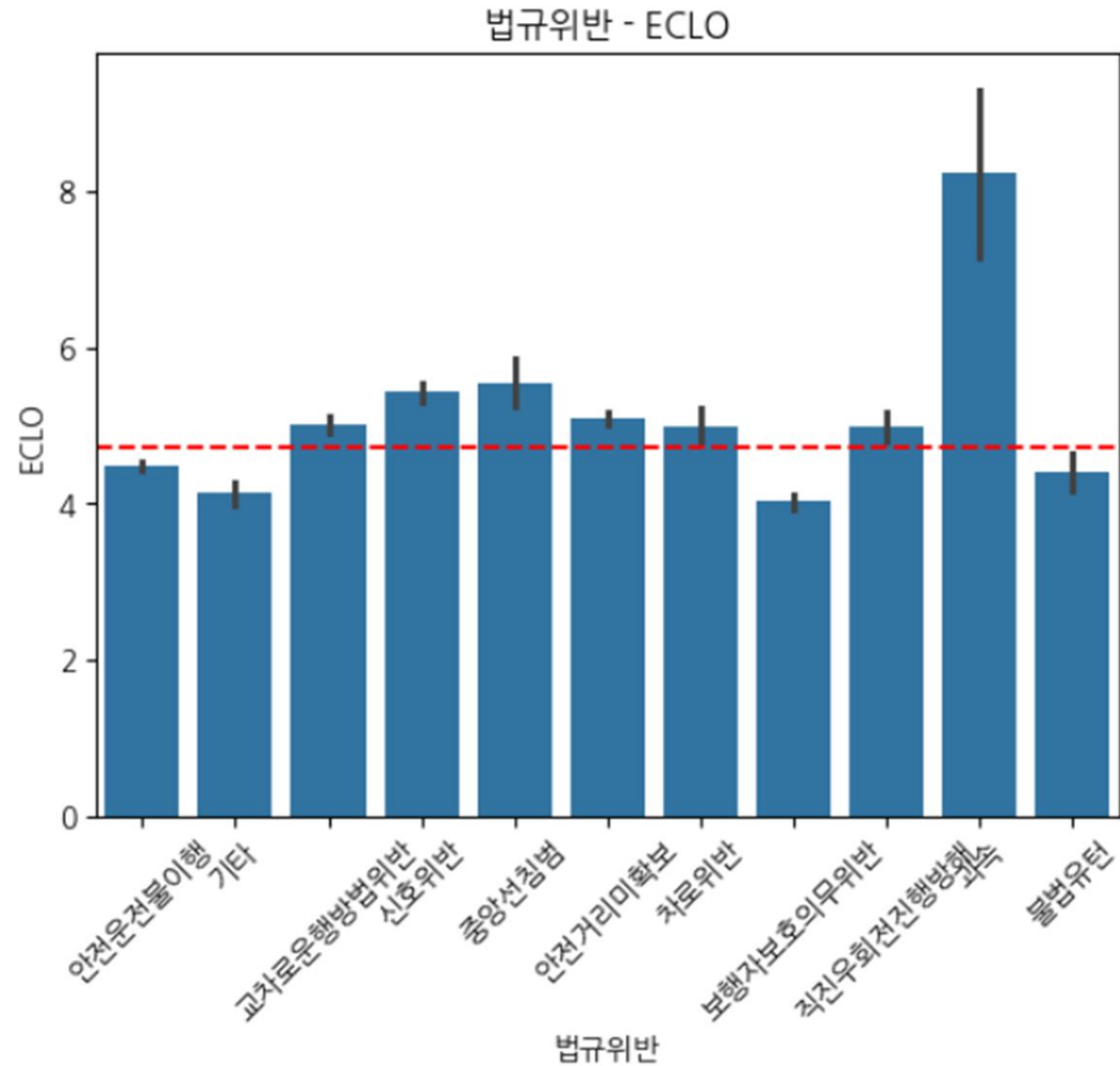
#1-4. EDA(지역)

1) '동' 별 평균 ECLO

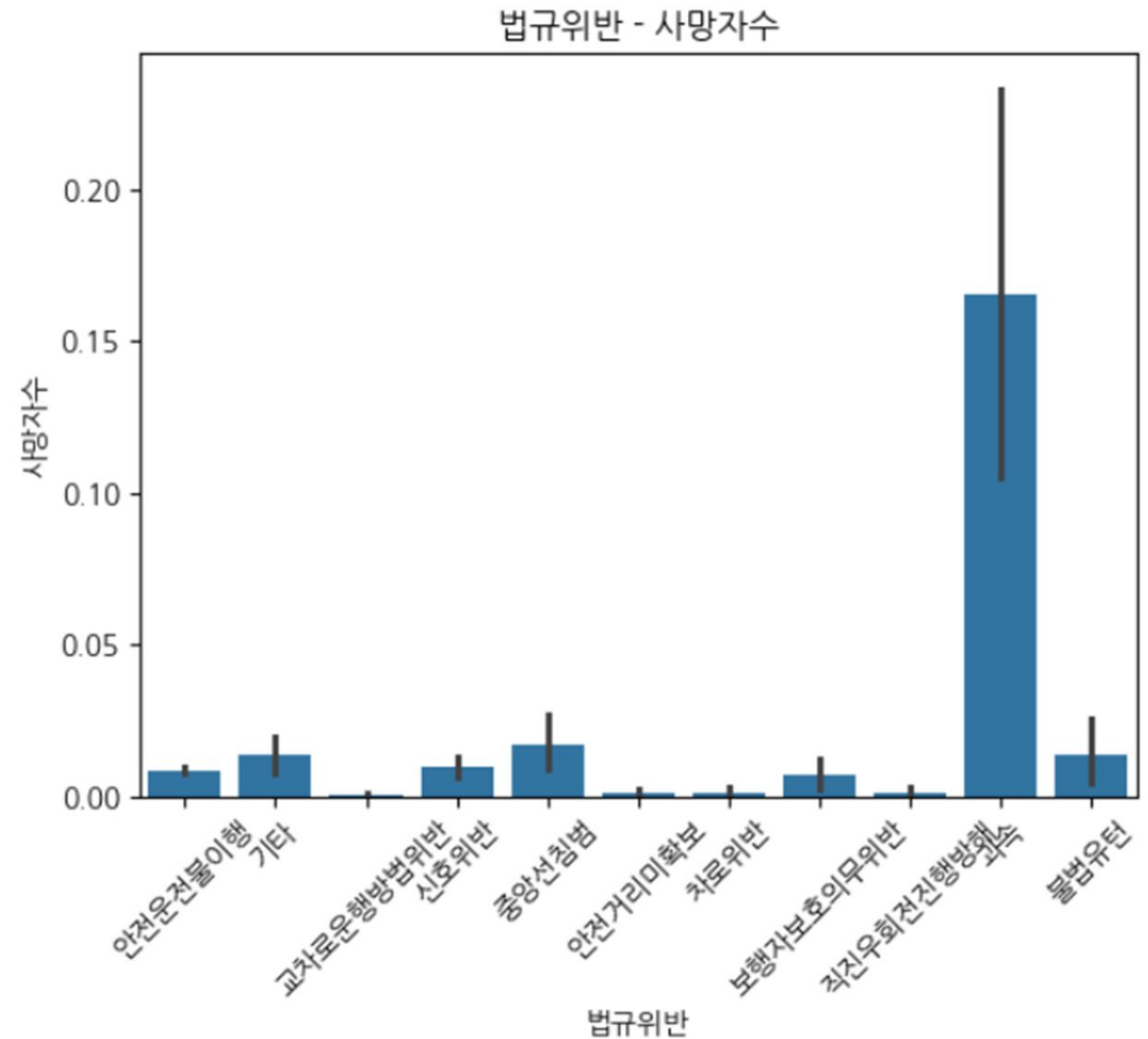


#1-5. EDA(법규위반)

1) 법규 위반의 ECLO 값

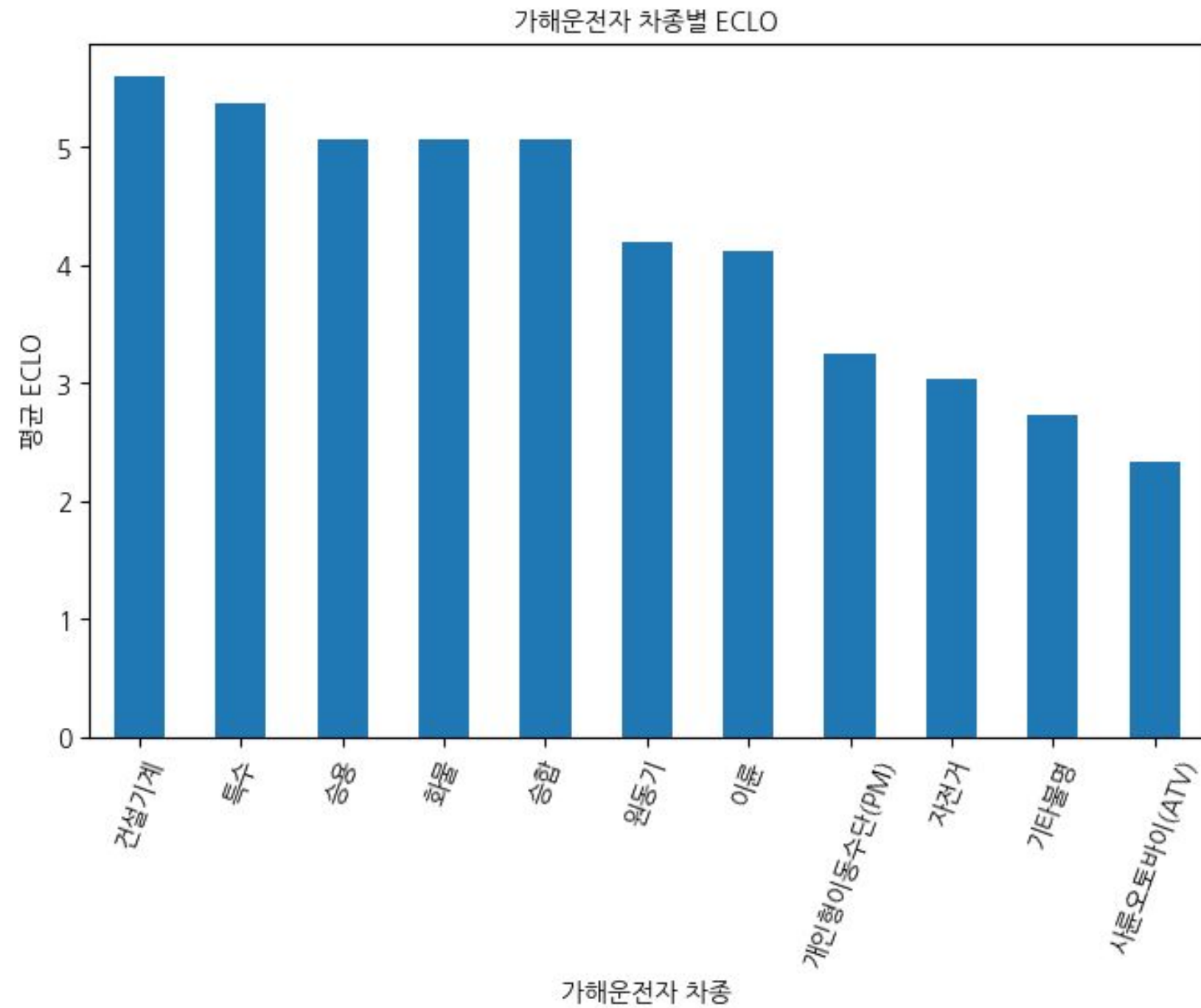


2) 법규 위반의 사망자수 확인

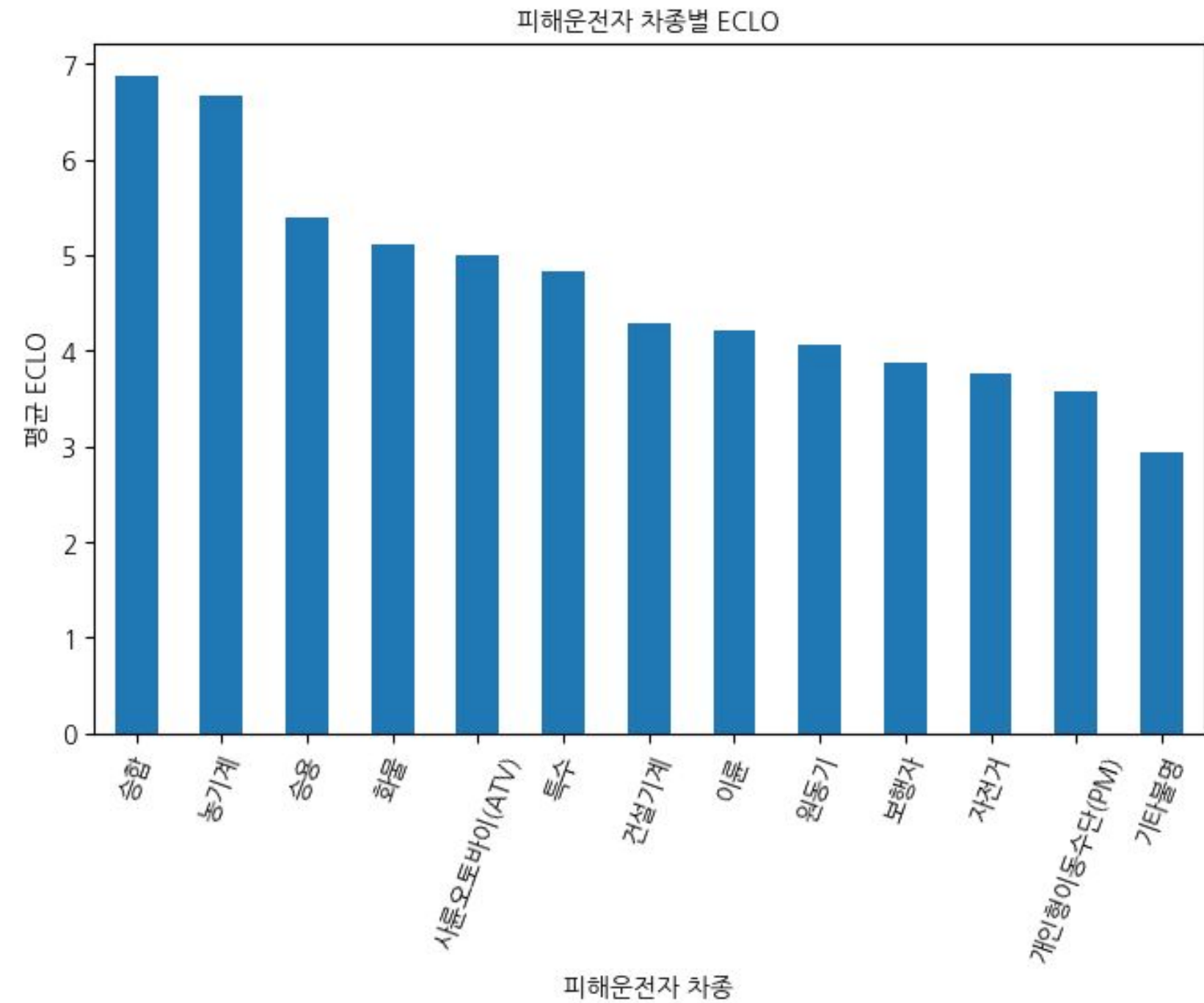


#1-6. EDA(가해/피해운전자)

1) 차종 - 가해운전자

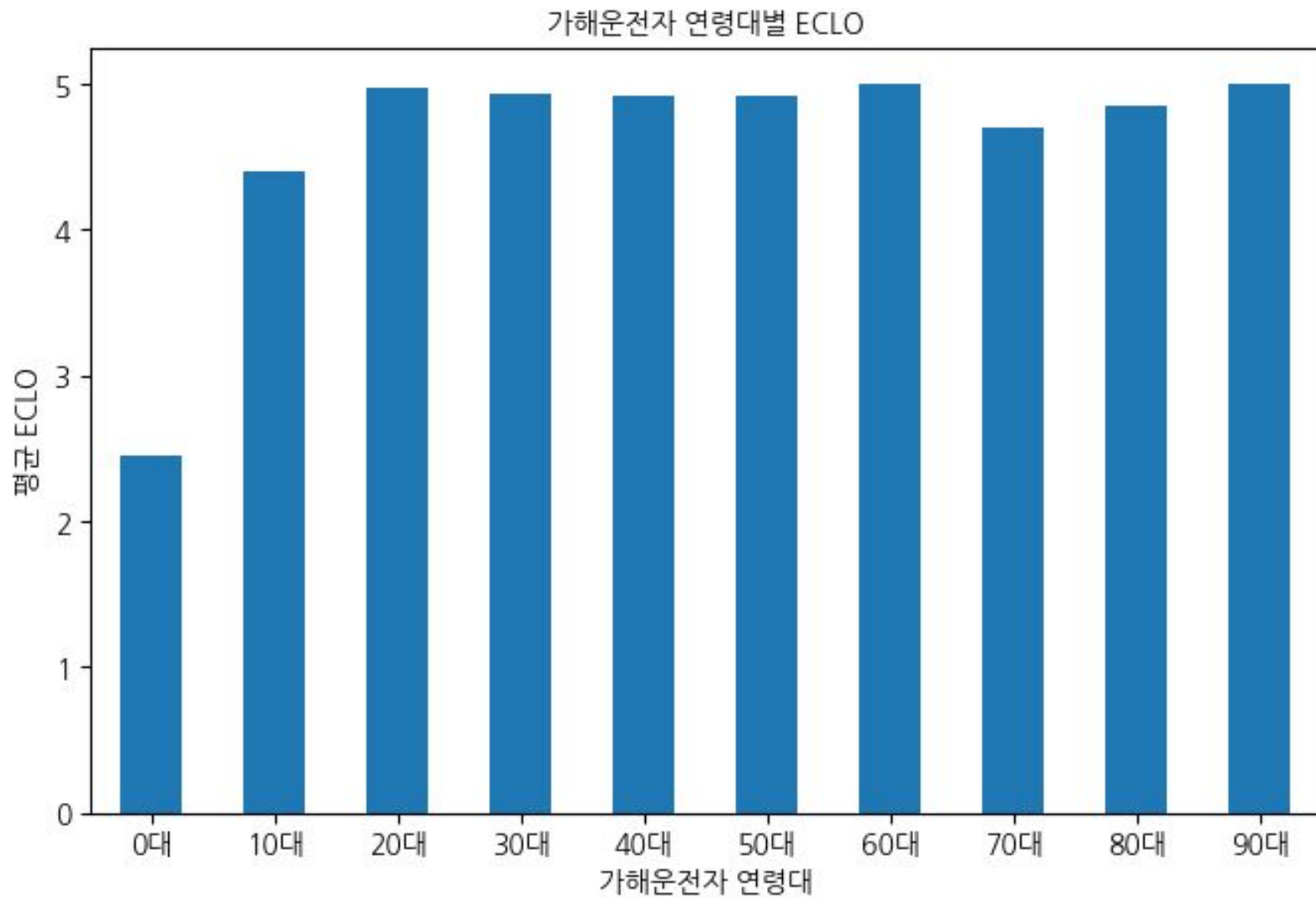


차종 - 피해운전자

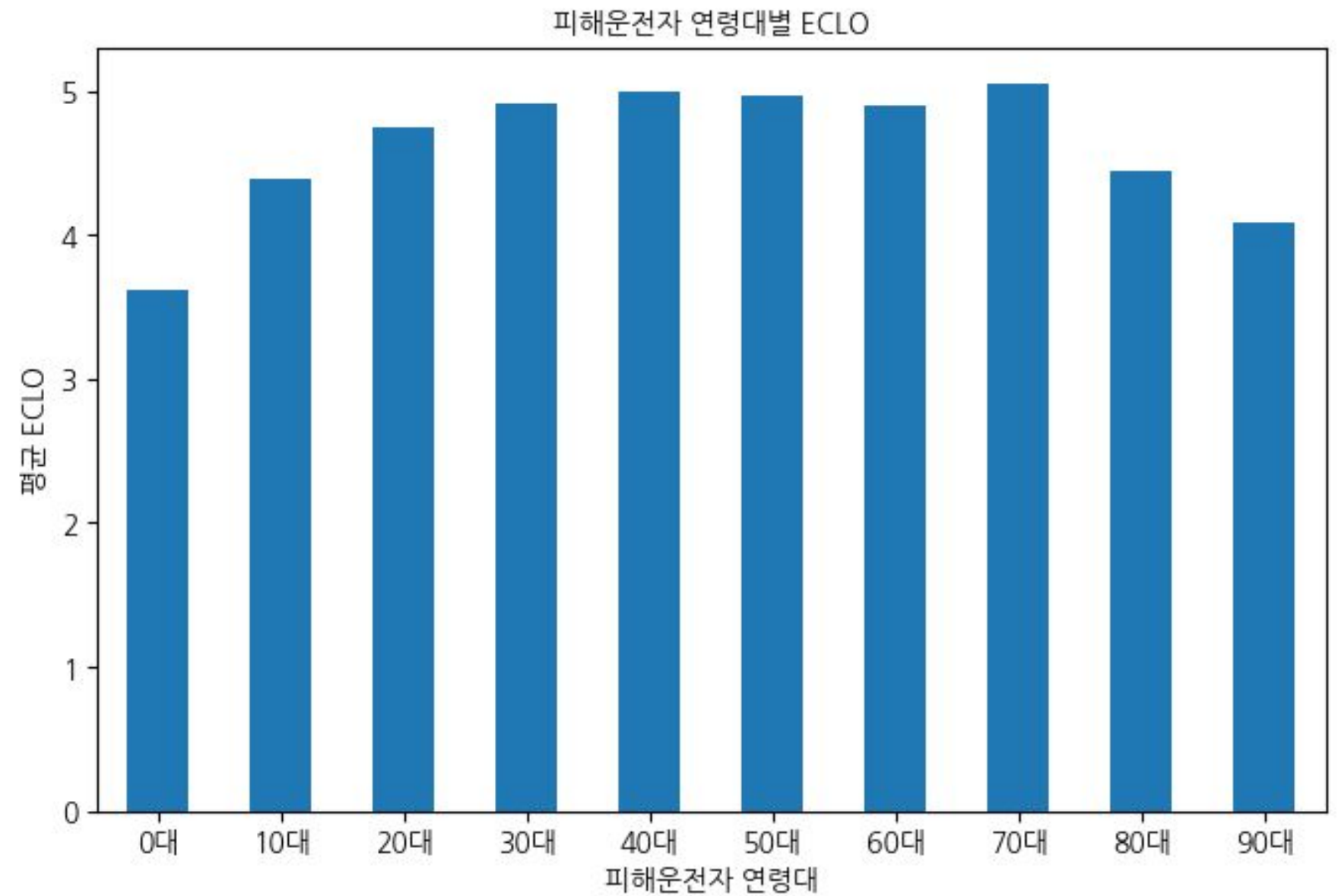


#1-6. EDA(가해/피해운전자)

2) 연령 - 가해운전자



연령 - 피해운전자



2. 전처리



#2-1. 사고시간 sin cos 변환

#sin cos 변환

##시간

```
train_df['sin_hour'] = np.sin(2 * np.pi * train_df['사고시간']/23.0)
train_df['cos_hour'] = np.cos(2 * np.pi * train_df['사고시간']/23.0)
test_df['sin_hour'] = np.sin(2 * np.pi * test_df['사고시간']/23.0)
test_df['cos_hour'] = np.cos(2 * np.pi * test_df['사고시간']/23.0)
```

##날짜

```
train_df['sin_date'] = -np.sin(2 * np.pi * (train_df['사고날짜'].dt.month + train_df['사고날짜'].dt.day / 31) / 12)
train_df['cos_date'] = -np.cos(2 * np.pi * (train_df['사고날짜'].dt.month + train_df['사고날짜'].dt.day / 31) / 12)
test_df['sin_date'] = -np.sin(2 * np.pi * (test_df['사고날짜'].dt.month + test_df['사고날짜'].dt.day / 31) / 12)
test_df['cos_date'] = -np.cos(2 * np.pi * (test_df['사고날짜'].dt.month + test_df['사고날짜'].dt.day / 31) / 12)
```

##월

```
train_df['sin_month'] = -np.sin(2 * np.pi * train_df['사고날짜'].dt.month / 12.0)
train_df['cos_month'] = -np.cos(2 * np.pi * train_df['사고날짜'].dt.month / 12.0)
test_df['sin_month'] = -np.sin(2 * np.pi * test_df['사고날짜'].dt.month / 12.0)
test_df['cos_month'] = -np.cos(2 * np.pi * test_df['사고날짜'].dt.month / 12.0)
```

#2-2. target 이상치 제거& 로그변환

1) target 변수 이상치 제거

✓ ECLO 이상치

```
[20] #ECLO 이상치 제거 (이상치의 기준은 0.001)  
  
train_df = train_df[train_df['ECLO'] < 27]
```

2) target 변수 로그변환

```
#ECLO 로그변환  
  
train_df['ECLO_log'] = np.log1p(train_df['ECLO'])
```


#2-3. 법규위반 가중치 생성

① 위반 법규별 평균 위험도(ECL0) 계산

```
illegal_dangerous = train_df[['법규위반', 'ECL0']].groupby('법규위반').mean()  
illegal_dangerous.columns = ['illegal_dangerous']
```

② 지역별 법규 위반 횟수 카운트

```
illegal_count = train_df[['도시', '구', '법규위반']]  
illegal_count['cnt'] = 1  
illegal_count = illegal_count.groupby(['도시', '구', '법규위반']).count()
```

③ 위반 법규와 위험도 곱하기

	도시	구	법규위반	cnt	illegal_dangerous	multiply
0	대구광역시	남구	과속	13	7.803030	101.439394
1	대구광역시	남구	교차로운행방법위반	139	5.011641	696.618043
2	대구광역시	남구	기타	81	4.084648	330.856528
3	대구광역시	남구	보행자보호의무위반	94	4.025130	378.362218
4	대구광역시	남구	불법유턴	24	4.398126	105.555035

#2-3. 법규위반 가중치 생성

④ ‘구’ 별로 총 위험도 합산 & 평균 위험도 계산

	도시	구	cnt	multiply
0	대구광역시	남구	2557	11930.103329
1	대구광역시	달서구	9886	46620.883695
2	대구광역시	달성군	2762	12822.497766
3	대구광역시	동구	5333	25078.206880
4	대구광역시	북구	6644	31240.122608
5	대구광역시	서구	3485	16336.308714
6	대구광역시	수성구	6698	31132.920177
7	대구광역시	중구	2194	10297.956831

	도시	구	illegal_dangerous
0	대구광역시	남구	4.665664
1	대구광역시	달서구	4.715849
2	대구광역시	달성군	4.642468
3	대구광역시	동구	4.702458
4	대구광역시	북구	4.702005
5	대구광역시	서구	4.687607
6	대구광역시	수성구	4.648092
7	대구광역시	중구	4.693690

⑤ 칼럼 추가

log	illegal_dangerous	ride_danger
759	4.693690	4.610
294	4.715849	4.716
294	4.648092	4.713
759	4.702005	4.587
294	4.702458	4.660

#2-4. 지역별 차종 위험도

① 차종별 평균 위험도(ECLO) 계산

```
ride_dangerous = train_df[['가해운전자 차종', 'ECLO']].groupby('가해운전자 차종').mean()  
ride_dangerous.columns = ['ride_dangerous']
```

② '동' 별 사고 차종 건수 카운트

```
ride_count = train_df[['도시', '구', '동', '가해운전자 차종']]  
ride_count['cnt'] = 1  
ride_count = ride_count.groupby(['도시', '구', '동', '가해운전자 차종']).count()  
ride_count.reset_index(inplace=True)
```

③ 차종별 건수와 위험도 곱하기

	도시	구	동	가해운전자 차종	cnt	ride_dangerous_off	multiply
0	대구광역시	남구	대명동	개인형이동수단(PM)	5	3.354651	16.773256
1	대구광역시	남구	대명동	건설기계	3	5.282353	15.847059
2	대구광역시	남구	대명동	기타불명	53	2.734151	144.910020
3	대구광역시	남구	대명동	승용	1204	4.878112	5873.246701
4	대구광역시	남구	대명동	승합	56	4.701972	263.310446
...

#2-4. 지역별 차종 위험도

④ ‘동’ 별 위험도 합산

	도시	구	동	cnt	multiply
0	대구광역시	남구	대명동	1748	8157.556732
1	대구광역시	남구	봉덕동	624	2940.244527
2	대구광역시	남구	이천동	185	860.804495
3	대구광역시	달서구	갈산동	110	525.289890
4	대구광역시	달서구	감삼동	678	3197.856712
...

⑤ ‘동’ 별 평균 위험도 계산

	도시	구	동	ride_dangerous_off
0	대구광역시	남구	대명동	4.666794
1	대구광역시	남구	봉덕동	4.711930
2	대구광역시	남구	이천동	4.652997
3	대구광역시	달서구	갈산동	4.775363
4	대구광역시	달서구	감삼동	4.716603
...

⑥ 파생변수 생성

	ID	요일	기상상태	도로형태	ride_dangerous_off	ride_dangerous_vic
0	ACCIDENT_00000	화요일	맑음	단일로 - 기타	4.610074	4.476552
1	ACCIDENT_00001	화요일	흐림	단일로 - 기타	4.716603	4.702283
2	ACCIDENT_00002	화요일	맑음	단일로 - 기타	4.713235	4.752123
3	ACCIDENT_00003	화요일	맑음	단일로 - 기타	4.587271	4.657062
4	ACCIDENT_00004	화요일	맑음	단일로 - 기타	4.660858	4.709142
...
39554	ACCIDENT_39604	금요일	맑음	교차로 - 교차로안	4.645028	4.529080
39555	ACCIDENT_39605	금요일	맑음	단일로 - 기타	4.662823	4.617467
39556	ACCIDENT_39606	금요일	맑음	교차로 - 교차로안	4.710121	4.739579
39557	ACCIDENT_39607	금요일	맑음	기타 - 기타	4.779813	4.967710
39558	ACCIDENT_39608	금요일	맑음	단일로 - 지하차도(도로)내	4.597156	4.610569

- ‘피해운전자 차종’도 1~5 과정 반복하여 추가

#2-5. 지역별 운전자 평균 연령

① '연령' 데이터 형식 변환

```
def age_transform(x):  
    try:  
        ret = int(x.split('세')[0])  
    except:  
        ret = np.NaN  
  
    return ret  
  
train_df['가해운전자 연령'] = train_df['가해운전자 연령'].apply(lambda x: age_transform(x))  
train_df['피해운전자 연령'] = train_df['피해운전자 연령'].apply(lambda x: age_transform(x))
```

② 지역별 평균 연령 계산

			가해운전자 평균연령	피해운전자 평균연령
도시	구	동		
대구광역시	남구	대명동	47.906949	44.158265
		봉덕동	48.606260	46.403624
		이천동	49.045455	48.044199
	달서구	갈산동	48.416667	46.247706
		감삼동	47.176383	42.906767


```
age_mean = train_df[['도시', '구', '동', '가해운전자 연령', '피해운전자 연령']].groupby(['도시', '구', '동']).mean()  
age_mean.columns = ['가해운전자 평균연령', '피해운전자 평균연령']
```

● 기존 데이터 형식

	가해운전자 연령	피해운전자 연령
0	51세	70세
1	39세	61세
2	70세	38세
3	49세	36세
4	30세	52세
...

③ 파생변수 생성

ID	요일	기상상태	가해운전자 평균연령	피해운전자 평균연령
ACCIDENT_00000	화요일	맑음	52.714286	48.015504
ACCIDENT_00001	화요일	흐림	47.176383	42.906767
ACCIDENT_00002	화요일	맑음	47.039384	43.267227
ACCIDENT_00003	화요일	맑음	46.894089	42.943902
ACCIDENT_00004	화요일	맑음	49.426415	47.400254

#2-6. 범주형 변수 인코딩

```
#categorical 변수 인코딩
#선형회귀와 같은 알고리즘이 아니라, 트리계열의 ml 알고리즘을 사용할 예정이므로 레이블 인코딩 방식을 채택.

#패키지 불러오기
from sklearn.preprocessing import LabelEncoder

#인코딩 함수를 통해 한번에 인코딩 시행
str_col = ['요일', '기상상태', '도로형태', '노면상태', '사고유형', '사고유형 - 세부분류', '법규위반',
           '가해운전자 차종', '가해운전자 성별', '가해운전자 연령', '가해운전자 상해정도', '피해운전자 차종',
           '피해운전자 성별', '피해운전자 연령', '피해운전자 상해정도', '계절', '도시', '구', '동']

for i in str_col:
    le = LabelEncoder()
    le = le.fit(train_df[i])
    train_df[i] = le.transform(train_df[i])
```

	ID	요일	기상상태	도로형태	노면상태	사고유형	사고유형 - 세부분류	법규위반
0	ACCIDENT_00000	6	2	6	0	0	2	7
1	ACCIDENT_00001	6	5	6	0	0	5	2
2	ACCIDENT_00002	6	2	6	0	0	9	7
3	ACCIDENT_00003	6	2	6	0	1	10	7
4	ACCIDENT_00004	6	2	6	0	1	10	7
...
33633	ACCIDENT_33677	6	3	1	4	1	11	1
33634	ACCIDENT_33678	6	2	6	4	1	11	2
33635	ACCIDENT_33679	6	2	6	0	1	1	5
33636	ACCIDENT_33680	6	2	1	0	1	11	7
33637	ACCIDENT_33681	6	5	1	4	1	11	1

3. 모델링



#3-1. LightGBM

1) GridSearchCV를 이용한 하이퍼 파라미터 튜닝

```
lgbm_parmas = {'learning_rate':[0.01], 'n_estimators':[1000],  
               'max_depth':[3,4,5,7,10],  
               'num_leaves':[10,20,30,40],  
               'min_child_samples':[50,100,200]}  
lgbm_reg = LGBMRegressor(metric='rmse', verbose=-1)  
best_lgbm = print_best_params(lgbm_reg, lgbm_parmas)
```

2) 최적의 하이퍼 파라미터 적용

```
lgbm_tune = LGBMRegressor(learning_rate=0.01,  
                           n_estimators=1000,  
                           max_depth=7,  
                           num_leaves=10,  
                           min_child_samples=200,  
                           metric='rmse',  
                           verbose=-1)
```

3) 모델 학습 및 예측, 평가

```
1 lgbm_tune.fit(X_train, y_train)  
2 lgbm_tune_pred = lgbm_tune.predict(X_test)  
3  
4 get_model_cv_prediction(lgbm_tune, X_features_drop, y_target_log)
```

```
##### LGBMRegressor #####  
5 교차 검증의 평균 RMSE : 0.4399919
```

- 5 교차 검증 평균 RMSE = 0.4399

#3-2. XGBoost

1) 최적의 파라미터 찾기

```
#최적의 파라미터
from sklearn.model_selection import GridSearchCV

# 하이퍼파라미터 그리드 정의
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6, 7, 8],
    'subsample' : [0.8, 0.9, 1.0],
    'colsample_bytree' : [0.8, 0.9, 1.0],
    'min_child_weight' : [10, 50, 100]
}

# GridSearchCV 객체 생성
grid_search = GridSearchCV(estimator=xgb_reg, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5)

# 모델 피팅 및 최적 파라미터 확인
grid_search.fit(X_features, y_target)
best_params = grid_search.best_params_
print("최적의 파라미터:", best_params)
```

최적의 파라미터: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 4, 'min_child_weight': 10, 'n_estimators': 100, 'subsample': 0.9}

#3-2. XGBoost

2) 모델 적용

```
xgb_reg2 = XGBRegressor(  
    n_estimators = 100,  
    max_depth=4,  
    learning_rate=0.1,  
    subsample=0.9,  
    colsample_bytree=0.8,  
    random_state=156,  
    min_child_weight=10,  
    objective='reg:squarederror',  
    eval_metric='rmse')
```

```
xgb_reg2.fit(X_train, y_train)
```

```
get_model_cv_prediction(xgb_reg2, X_features, y_target_log)
```

```
##### XGBRegressor #####  
5 교차 검증의 평균 RMSE : 0.4403002
```

- 5 교차 검증 평균 RMSE = 0.4403

#3-3. Catboost

1) GridSearchCV 및 파라미터 조정

```
param_grid = {  
    'n_estimators': [50, 100, 150, 200, 250],  
    'depth': [3, 4, 5],  
    'learning_rate': [0.001, 0.005, 0.01, 0.05, 0.1],  
    'min_child_samples': [5, 10, 20, 40],  
}
```

2) 모델 학습 및 결과

```
cat = catboost.CatBoostRegressor(n_estimators=200, learning_rate=0.01, min_child_samples=15)
```

```
cat.fit(X_train, y_train)
```

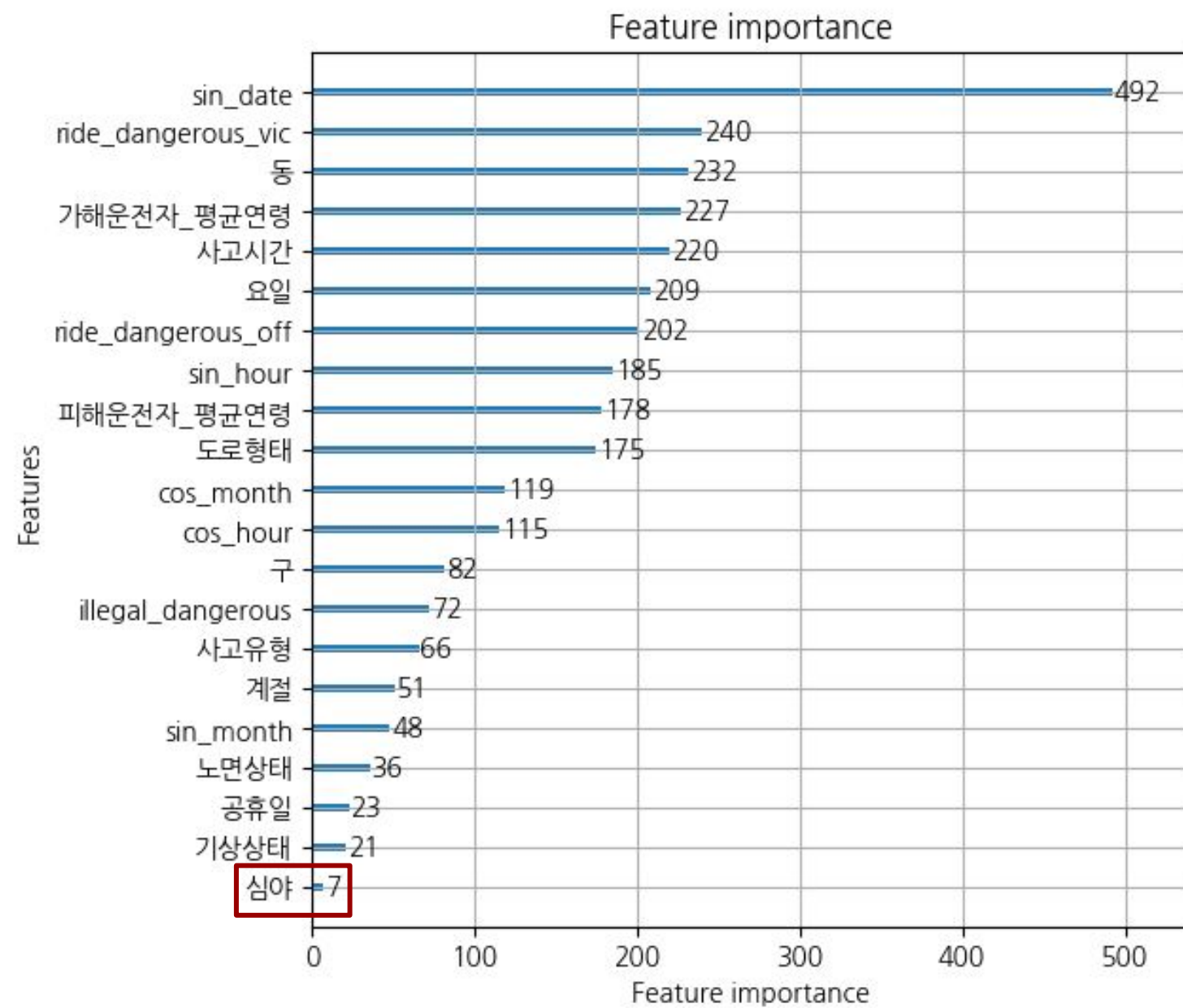
```
##### CatBoostRegressor #####  
5 교차 검증의 평균 RMSE : 0.4412461
```

- 5 교차 검증 평균 RMSE = 0.4412

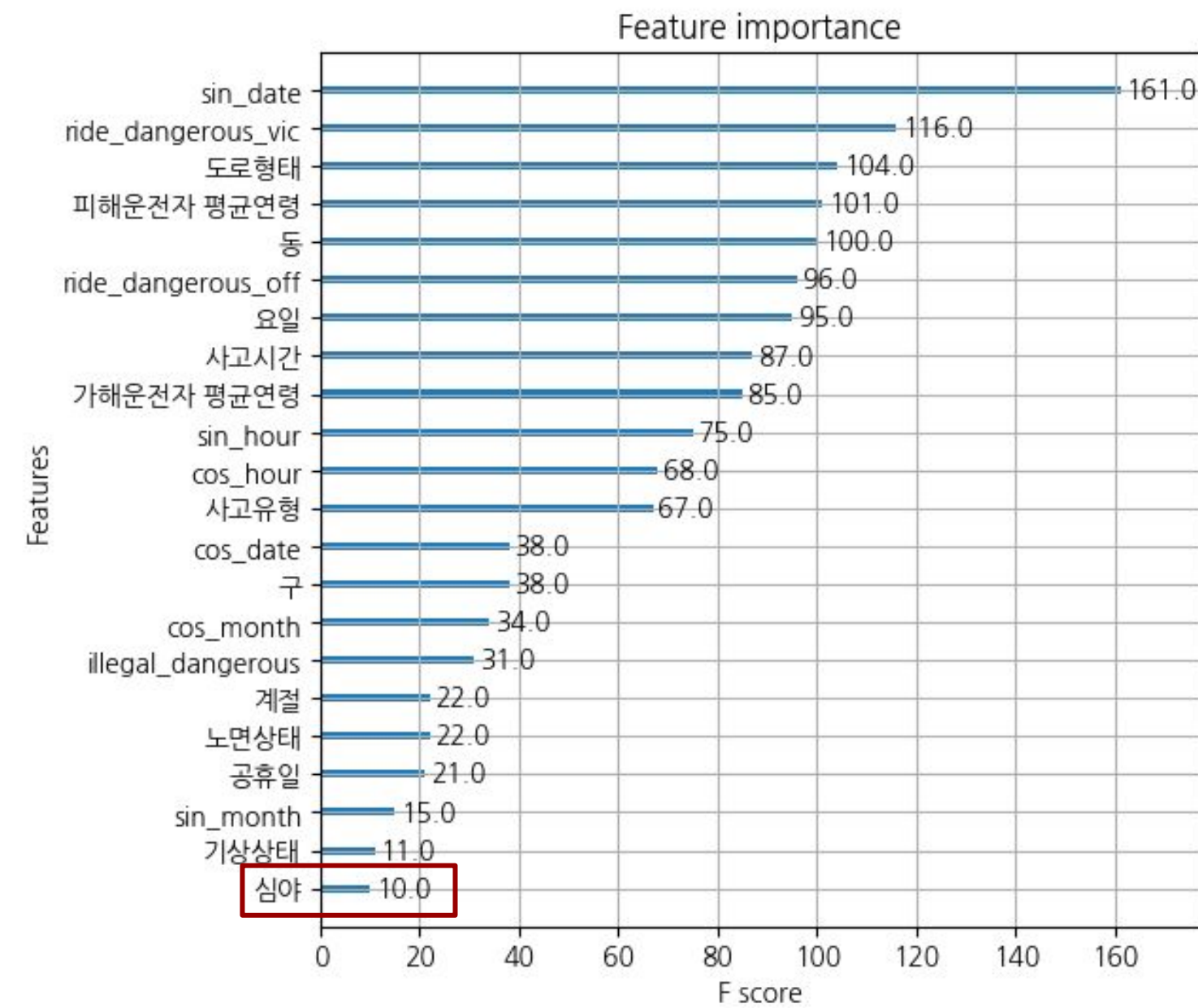
#3-4. 피쳐 선택

○ 피쳐 중요도

● lightgbm



● xgboost



- 피쳐 중요도 가장 낮은 '심야' 컬럼 제거 -> 평균 RMSE 감소

#3-5. 최종 혼합 모델

1) 개별 모델 학습 결과

```
1 # lgbm
2 pred_fin_lgbm_log = lgbm_tune.predict(test_ftr_drop)
3 pred_fin_lgbm_log

array([1.62640515, 1.58080767, 1.79323573, ..., 1.77069073, 1.65024069,
       1.74132125])
```

```
1 # xgb
2 pred_fin_xgb_log = xgb_reg2.predict(test_ftr_drop)
3 pred_fin_xgb_log

array([1.5810366, 1.5382851, 1.8092607, ..., 1.7175264, 1.6830648,
       1.7578423], dtype=float32)
```

```
1 # Catboost
2 pred_fin_cat_log = cat.predict(test_ftr_drop)
3 pred_fin_cat_log

array([1.59052833, 1.5292949, 1.71556632, ..., 1.69912029, 1.68260186,
       1.68685349])
```

2) 예측 결과 혼합

```
1 # 가중치 부여 및 합산
2 pred_fin_esb_log = 0.5*pred_fin_lgbm_log + 0.1*pred_fin_cat_log + 0.4*pred_fin_xgb_log
3
4 # 최종 예측값
5 pred_fin_esb = np.expm1(pred_fin_esb_log)
6 pred_fin_esb

array([3.97872631, 3.65995479, 4.82827031, ..., 4.73159321, 4.28015345,
       4.55332341])
```

- **LGBM : XGB : Cat = 0.5 : 0.4 : 0.1**
- **혼합 모델 RMSE = 0.4366**
→ 개별 모델보다 약 0.003 이상 감소

4. 결론



4-1. 데이콘 제출결과

1) 최초 모델 (가장 저조) ~ 튜닝 과정

public점수	0.4286560473	0.4275560663	0.4275168232	0.4272138042	0.427477158	0.4274216672	0.4274498835
private점수	0.4294104223	0.4272384613	0.4270733701	0.4283956654	0.4271279781	0.4270006441	0.4269617008





- 하이퍼 파라미터 튜닝, 가중치 변경, 피쳐 선택 등

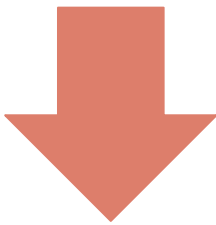
2) 최종 모델 (가장 탁월)



sub_esb_fin_541_new_drop_lxc.csv	2024-02-18 23:58:32	0.4273215333
피해 차종 추가, lxc 541, 심야 drop edit		0.4267510264

- private 점수: 0.4294 -> 0.4267로 향상

4-1. 데이콘 제출결과

272	으넵이		0.42939	5	2달 전
273	Jhyunye		0.4294	2	3달 전
274	김영혁		0.42945	12	3달 전
275	lkjjj		0.4295	1	3달 전



18	도로롱쓰		0.42674	4	2달 전
19	대구르르	  	0.42675	18	2달 전
20	yuldo		0.42675	44	2달 전
21	감굴쭈글	  	0.42676	50	3달 전

THANK YOU

