범죄 유형 분류 AI

해커톤 4조
- 이제이 202021931
- 성예담 202126162
- 김진찬 201820196

라이브러리 불러오기 & 랜덤시드 고정

■ 라이브러리 불러오기

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import random
import os
import matplotlib.pyplot as plt
import seaborn as sns

# 한글 폰트
from matplotlib import font_manager, rc
font_path = "./malgun.ttf"
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)

import scipy.linalg as la
from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

# !pip install catboost
import catboost
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# !pip install optuna
import optuna
from optuna import Trial, visualization
from optuna.samplers import TPESampler
```

■ 랜덤시드 고정

```
def seed_everything(seed):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)

seed_everything(42)
```

=> 시드를 42로 설정함으로써 프로그램을 실행할 때,

모든 실행에서 동일한 난수 시퀀스를 얻을 수 있도록 한다.

=> 이는 디버깅과 결과의 재현성을 보장하는 데 유용하다.

데 이 터 파 악

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84406 entries, 0 to 84405
Data columns (total 20 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        84406 non-null  object
 1   월         84406 non-null  int64
 2   요일        84406 non-null  object
 3   시간        84406 non-null  int64
 4   소관경찰서     84406 non-null  int64
 5   소관지역      84406 non-null  float64
 6   사건발생거리    84406 non-null  float64
 7   강수량(mm)  84406 non-null  float64
 8   강설량(mm)  84406 non-null  float64
 9   적설량(cm)  84406 non-null  float64
 10  풍향        84406 non-null  float64
 11  안개        84406 non-null  float64
 12  짙은안개      84406 non-null  float64
 13  번개        84406 non-null  float64
 14  진눈깨비      84406 non-null  float64
 15  서리        84406 non-null  float64
 16  연기/연무     84406 non-null  float64
 17  눈날림       84406 non-null  float64
 18  범죄발생지     84406 non-null  object
 19  TARGET    84406 non-null  int64
dtypes: float64(13), int64(4), object(3)
memory usage: 12.9+ MB
```

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17289 entries, 0 to 17288
Data columns (total 19 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        17289 non-null  object
 1   월         17289 non-null  int64
 2   요일        17289 non-null  object
 3   시간        17289 non-null  int64
 4   소관경찰서     17289 non-null  int64
 5   소관지역      17289 non-null  float64
 6   사건발생거리    17289 non-null  float64
 7   강수량(mm)  17289 non-null  float64
 8   강설량(mm)  17289 non-null  float64
 9   적설량(cm)  17289 non-null  float64
 10  풍향        17289 non-null  float64
 11  안개        17289 non-null  float64
 12  짙은안개      17289 non-null  float64
 13  번개        17289 non-null  float64
 14  진눈깨비      17289 non-null  float64
 15  서리        17289 non-null  float64
 16  연기/연무     17289 non-null  float64
 17  눈날림       17289 non-null  float64
 18  범죄발생지     17289 non-null  object
dtypes: float64(13), int64(3), object(3)
memory usage: 2.5+ MB
```

=> info()를 통해 결측치가 없는 것을 확인

Dtype	Column
실수형(float64)	소관지역, 사건발생거리, 강수량(mm), 강설량(mm), 적설량(cm), 풍향, 안개, 짙은안개, 번개, 진눈깨비, 서리, 연기/연무, 눈날림
문자형(object)	ID, 요일, 범죄발생지
정수형(int64)	월, 소관경찰서, TARGET(train data)

=> 실수형, 정수형, 문자형 등 다양한 데이터 타입 존재

가 설

EDA를 하기 전 세운 가설은 다음과 같다.

■ **가설 1**: 범죄발생지 유형을 보아 유동인구가 많은 시간대에 범죄가 많이 발생했을 것이다.

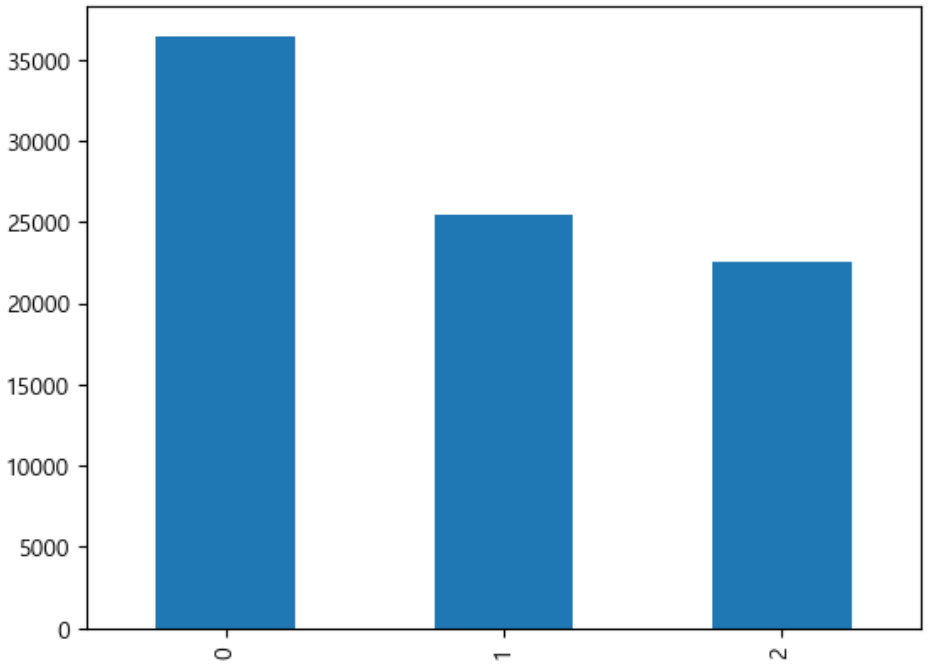
■ **가설 2**: 특정 범죄발생지에서 특정 범죄가 많이 발생했을 것이다.

(예를 들어 주자장의 경우, 빈 차를 노린 절도가 많이 발생했을 것이다.)

■ **가설 3**: 비나 눈이 오는 날 교통사고로 인한 상해가 많이 발생했을 것이다.

가. 타겟 변수의 분포 확인

타겟 변수가 고르게 분포되어 있는지 확인

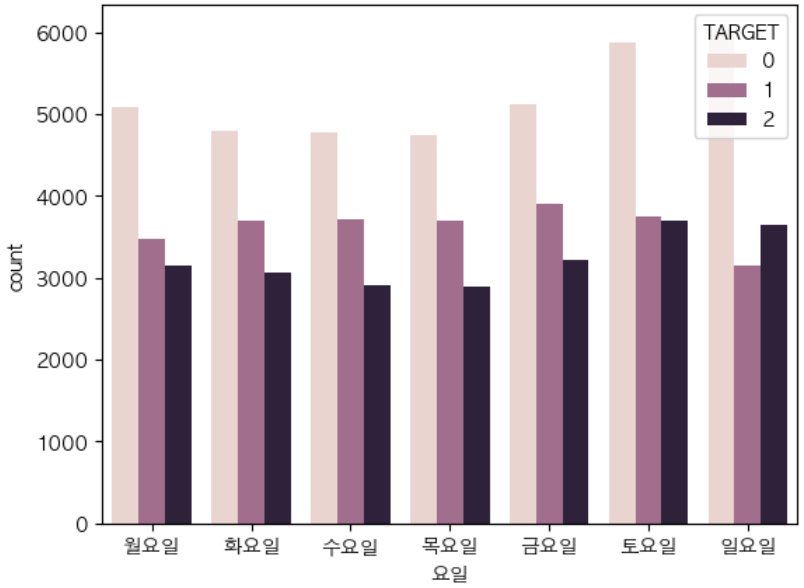



```
범죄 유형 비율 (%)
0    43.187688
1    30.089093
2    26.723219
Name: TARGET, dtype: float64
```

=> 타겟 변수에 불균형이 있는 것을 확인했다.

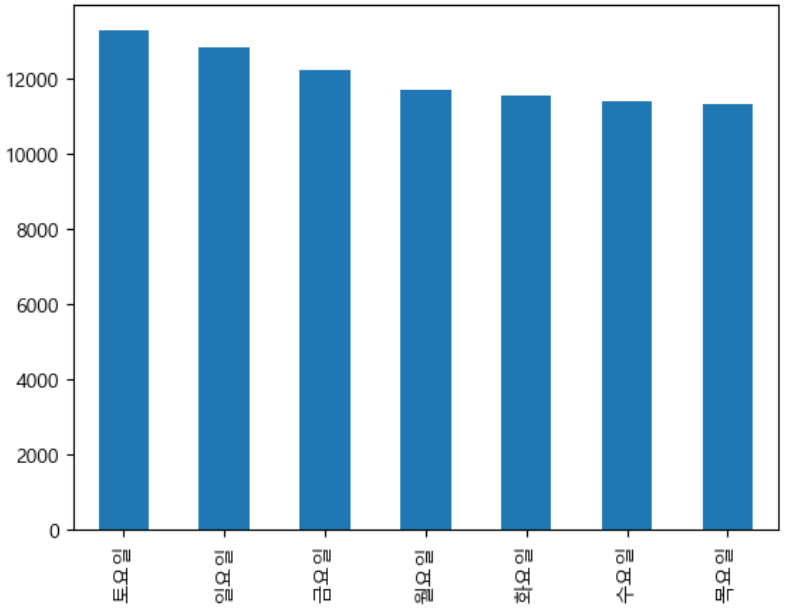
강도의 빈도가 가장 많았다.

나. 요일별 타겟 분포 확인



=> 월요일, 화요일, 수요일, 목요일, 금요일은 비슷한 추이로 타겟 값이 분포

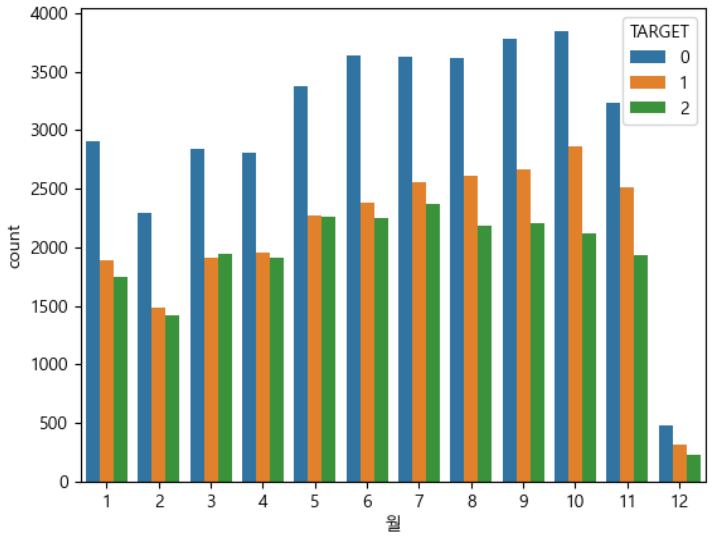
=> 토요일에 강도의 발생이 가장 많았으며, 일요일은 상해의 빈도 수가 많은 특이한 분포



=> 주말과 금요일에 범죄 발생 수가 가장 많음

=> 이는 유동인구가 많기 때문에 그런 것으로 예상

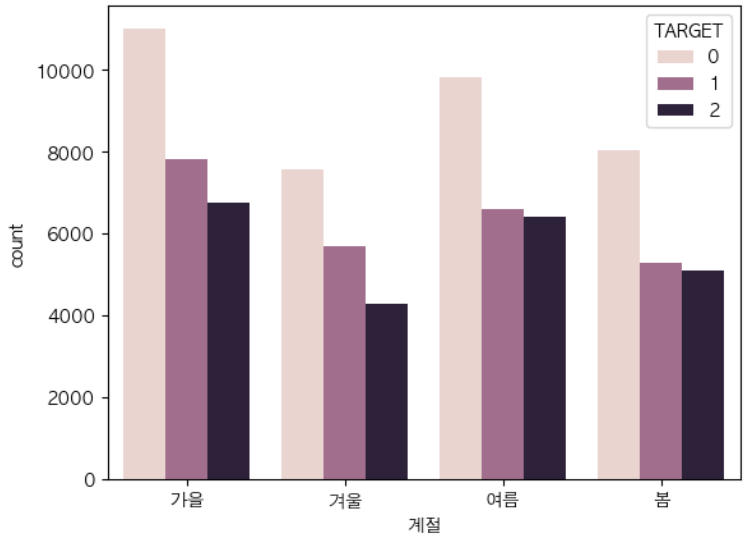
다. 월별 타겟 분포 확인



=> 12월 데이터의 수가 확연히 적다는 것을 확인

=> 데이터 수집 시기가 다음 해 연초라면?

직전인 12월 데이터가 제대로 수집되지 않았을 가능성



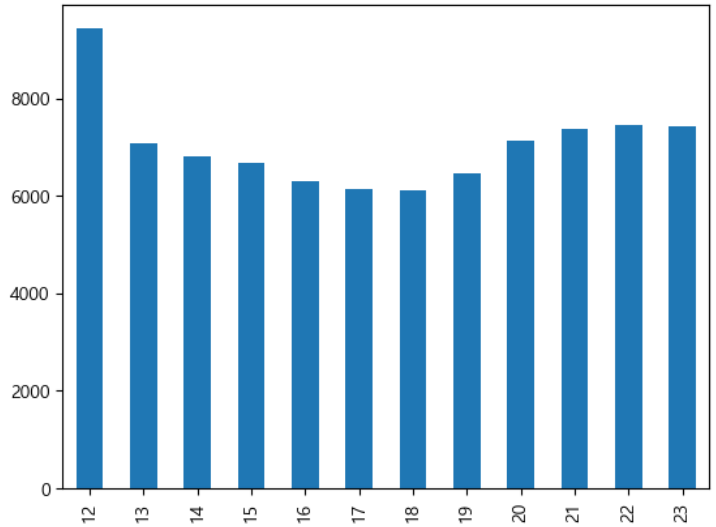
=> 가을에 범죄 발생량이 가장 많았으며,

강도의 수가 가장 많음

=> 겨울에 범죄 발생량이 가장 적었으며,

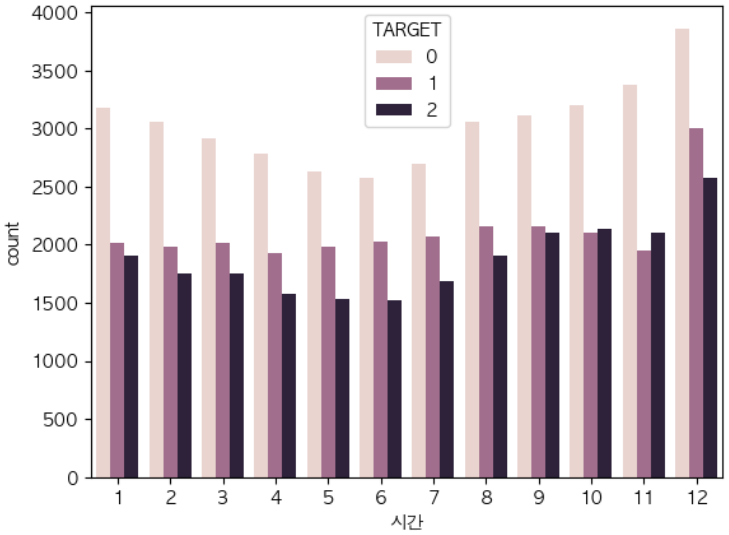
이는 12월 데이터가 적기 때문인 것으로 추측

라. 시간별 타겟 분포 확인



=> 시간별로 타겟 값의 분포를 확인해 본 결과,

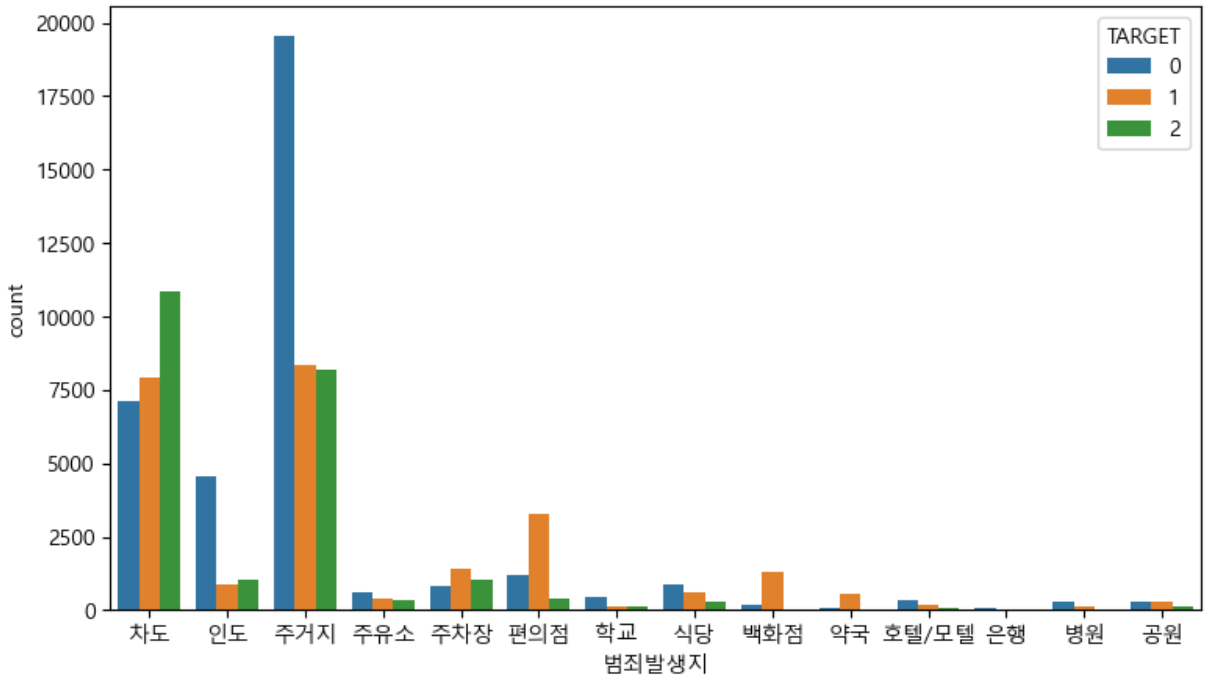
12시에 범죄가 가장 많이 일어나는 것으로 파악됨



=> 10, 11 시간대에는 상해가 절도보다

많이 일어나는 것을 알 수 있음

마. 장소에 따른 타겟 분포 확인



=> 편의점, 백화점, 약국 등 판매업소에서
절도가 특히 많이 일어남

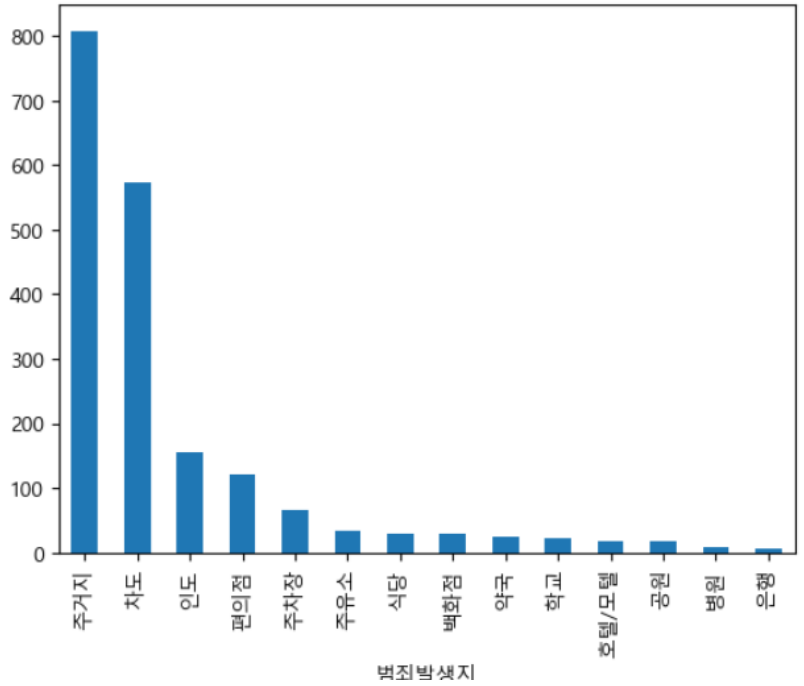
=> 주차장에서도 절도가 많이 일어나는데,
사람이 없는 빈 차가 많기 때문으로 예상

=> 차도에서 상해가 특히 많이 일어나는데,
이는 교통사고로 추정

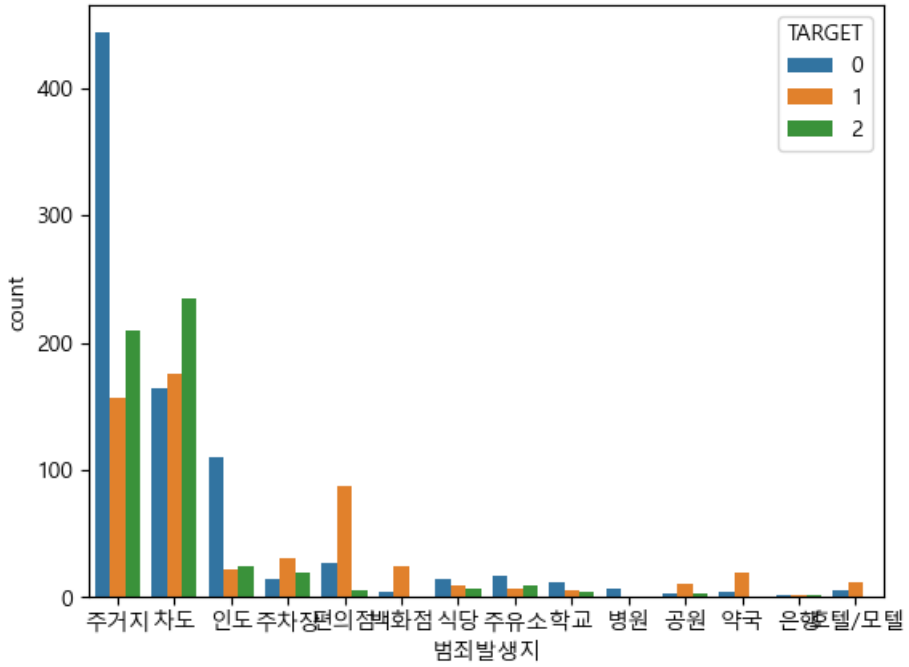
=> 주유소, 학교, 식당, 호텔/모텔, 은행, 병원, 공원은
데이터의 수가 매우 적으며 특이사항 없음

바. 날씨에 따른 타겟 분포 확인

1. 비가 많이 오는 날 (200mm 이상)



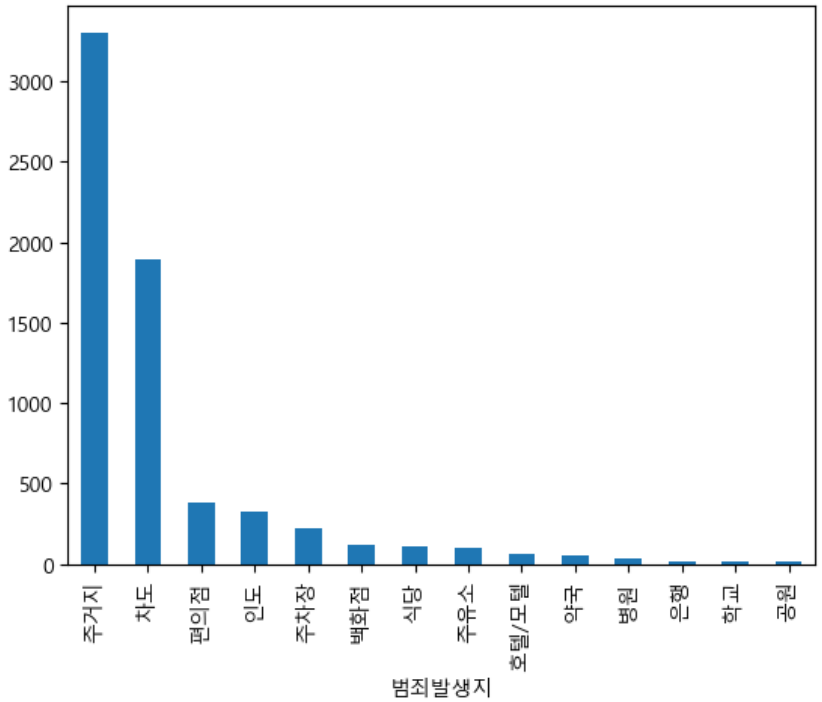
=> 비가 많이 오는 날 주거지, 차도, 인도, 편의점 순으로 범죄가 많이 일어나는 것을 알 수 있음



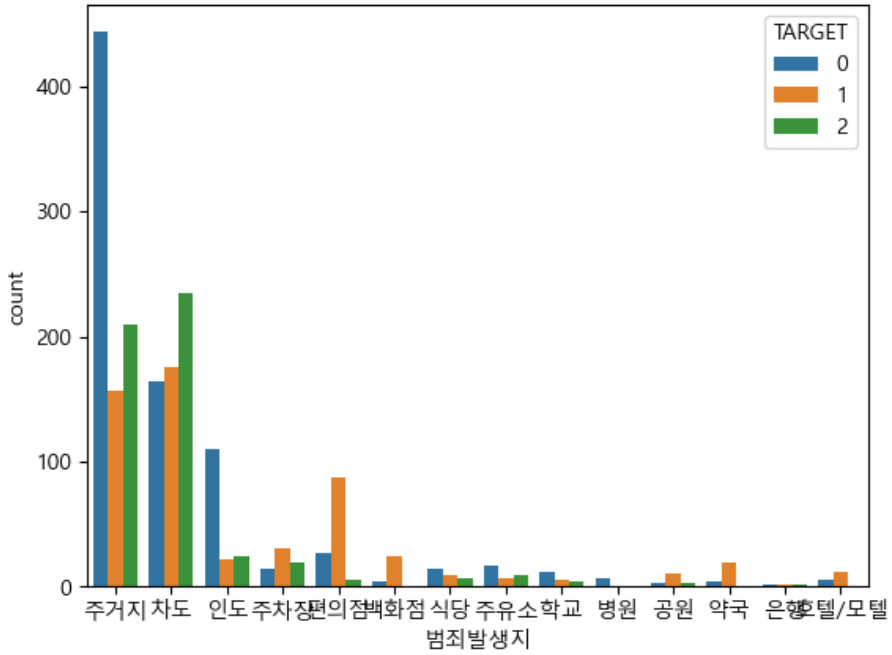
=> 비가 많이 오는 날 주거지의 경우 강도의 발생량이 가장 많음

바. 날씨에 따른 타겟 분포 확인

2. 눈이 오는 날



=> 주거지, 차도, 편의점, 인도, 주차장 순으로 범죄 발생 건수가 많았다.



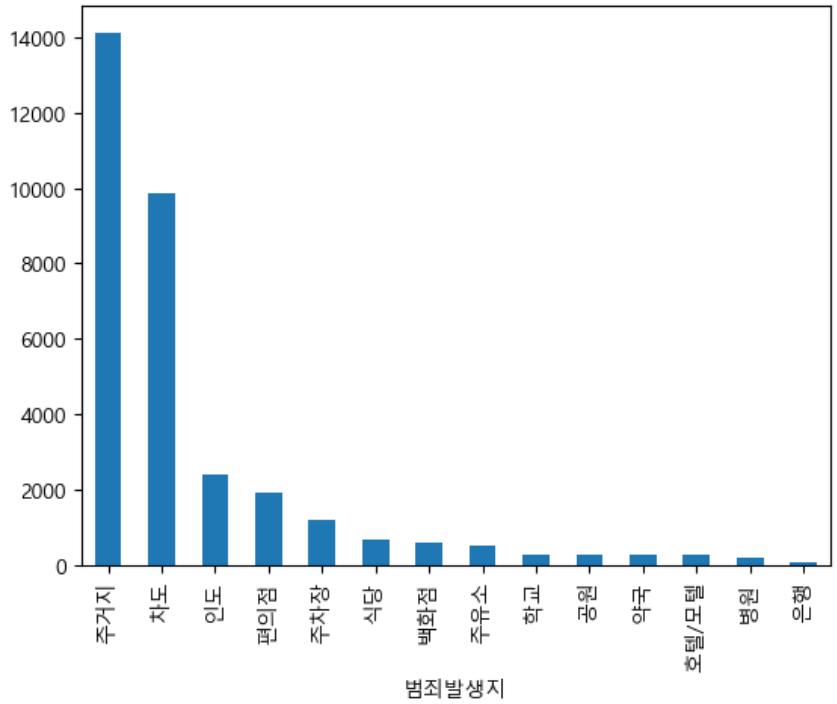
=> 주거지의 경우 강도가 가장 많았으며, 절도와 상해 수는 비슷

=> 차도의 경우 상해가 가장 많음

=> 편의점은 절도가, 인도는 강도의 발생량이 많음

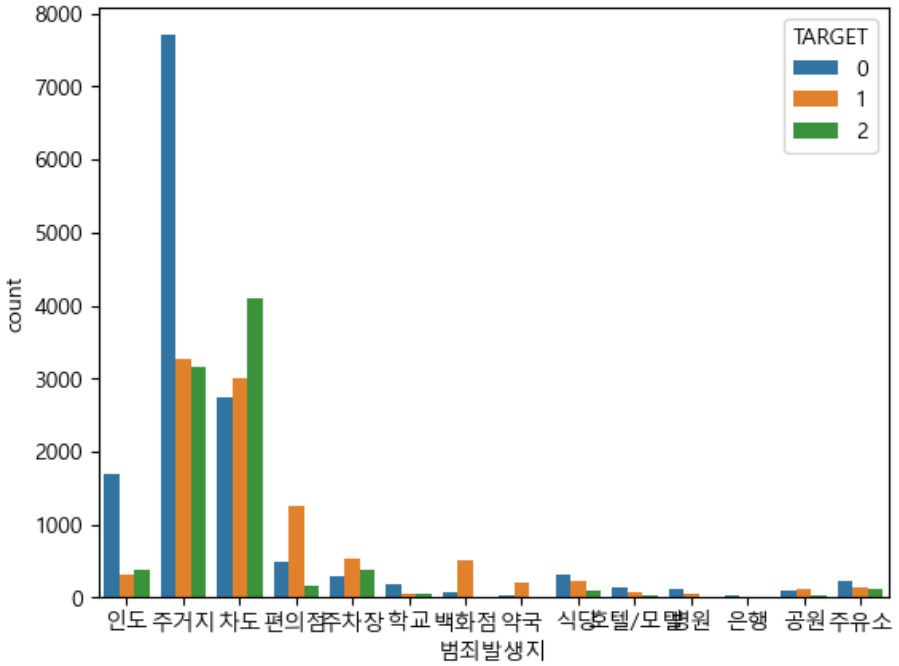
바. 날씨에 따른 타겟 분포 확인

3. 안개가 낀 날



=> 주거지, 차도, 인도, 편의점

순으로 범죄 발생 건수가 많음

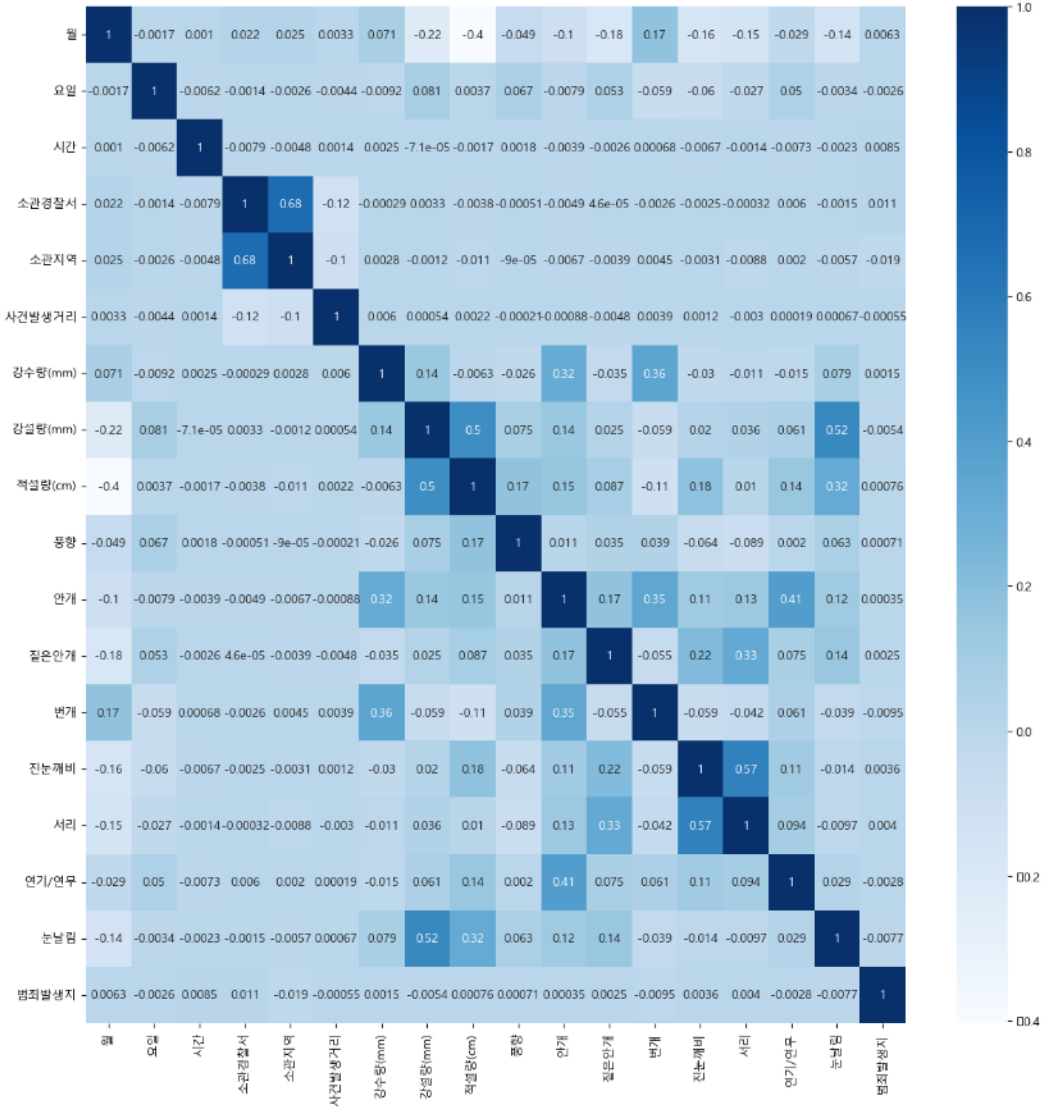


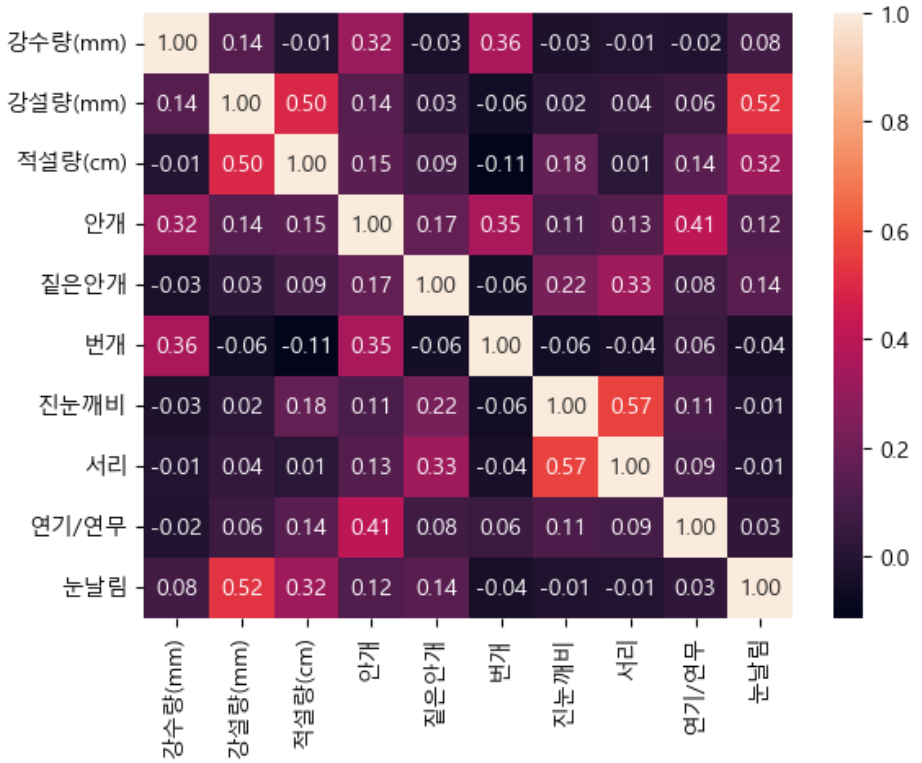
=> 주거지의 경우 강도가 가장 많았으며, 절도와 상해 수는 비슷

=> 차도의 경우 상해가 가장 많음

=> 편의점은 절도가, 인도는 강도의 발생량이 많음

인 사 이 트 도 출





=> 기상 정보 데이터만을 추출해 상관계수를 파악

=> 서리와 진눈깨비, 강설량과 눈날림, 적설량과 강설량 등의 경우 서로 비슷한 변수라 다중공선성의 문제가 생길 수 있음

=> 따라서 차원을 축소하거나 일부 변수를 제거하는 방향으로 전처리를 진행함

주성분 분석

```
train_df_weather = train_df.iloc[:, 6:-2]

scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(train_df_weather), columns=train_df_weather.columns)

result_corr = la.eig(x.corr(method='pearson'))
sorted(result_corr[0], reverse=True) # 고윳값 1 이상: 4개
```

```
[(2.211699323888107+0j),
 (1.7185971071441544+0j),
 (1.670690177389527+0j),
 (1.0604511591233734+0j),
 (0.8675007092644866+0j),
 (0.6291664626700502+0j),
 (0.6109480637372497+0j),
 (0.4563025945032726+0j),
 (0.43640672977953454+0j),
 (0.3382376725002512+0j)]
```

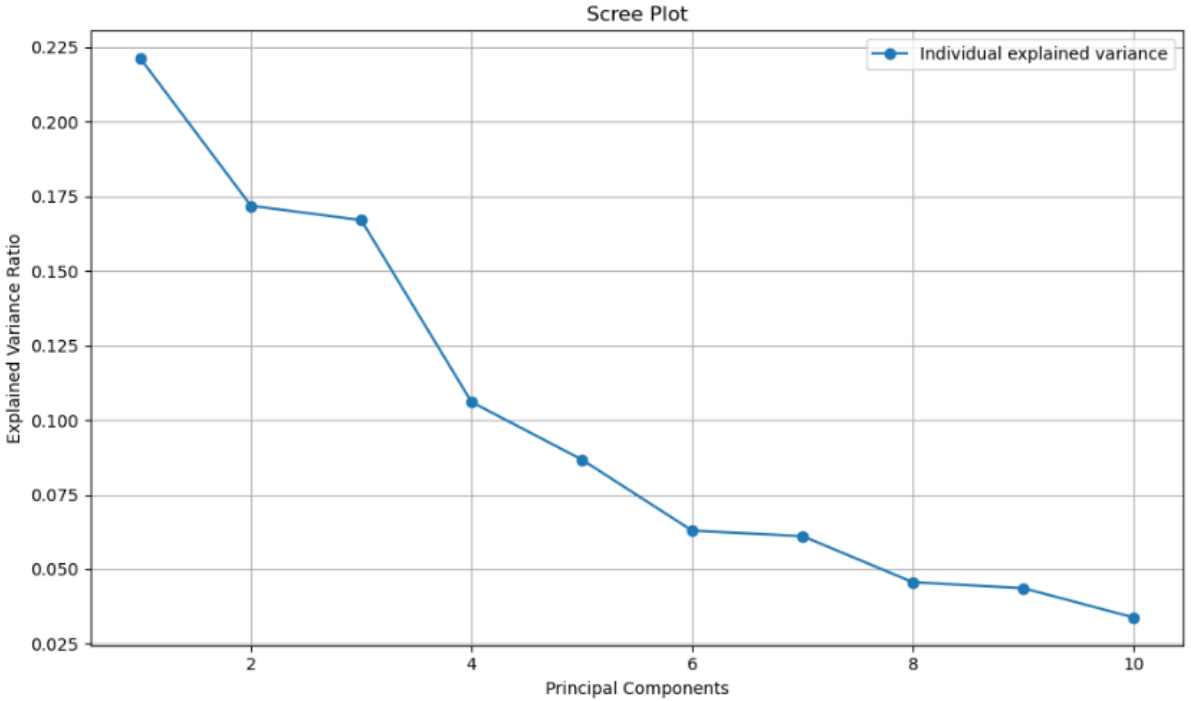
	Variance_percent	Cumulative_variance_percent
PC1	0.221170	0.221170
PC2	0.171860	0.393030
PC3	0.167069	0.560099
PC4	0.106045	0.666144
PC5	0.086750	0.752894
PC6	0.062917	0.815810
PC7	0.061095	0.876905
PC8	0.045630	0.922536
PC9	0.043641	0.966176
PC10	0.033824	1.000000

=> Eigen value(고윳값)을 확인해본 결과 1 이상인 변수는 4개였다.

=> 누적 기여율은 5개의 주성분을 사용했을 때 약 75%

=> 누적 기여율은 70% ~ 90%가 될 때 충분하다고 판단

주 성 분 분 석



=> 고윳값, 누적 기여율, Scree Plot을 모두 고려해

사용할 주성분의 개수를 5개로 결정

<Loading>

	PC1	PC2	PC3	PC4	PC5
강수량(mm)	0.251653	-0.396335	-0.509887	0.458962	-0.125921
강설량(mm)	0.627183	-0.405553	0.408644	0.124002	-0.124026
적설량(cm)	0.611565	-0.191060	0.410006	-0.146111	-0.258386
안개	0.577419	-0.151490	-0.572917	-0.229632	0.096957
짙은안개	0.417331	0.426579	-0.010591	0.131014	0.717190
번개	0.100418	-0.310087	-0.710748	0.193304	-0.023076
진눈깨비	0.448052	0.659107	-0.040687	0.152295	-0.393500
서리	0.428945	0.698675	-0.113968	0.249536	-0.125784
연기/연무	0.413192	0.048507	-0.284302	-0.768106	-0.015619
눈날림	0.554856	-0.366803	0.382921	0.174270	0.272573

■ PC1: 강설량(+), 적설량(+), 안개(+), 눈날림(+) → 눈이 오는 정도

■ PC2: 진눈깨비(+), 서리(+)

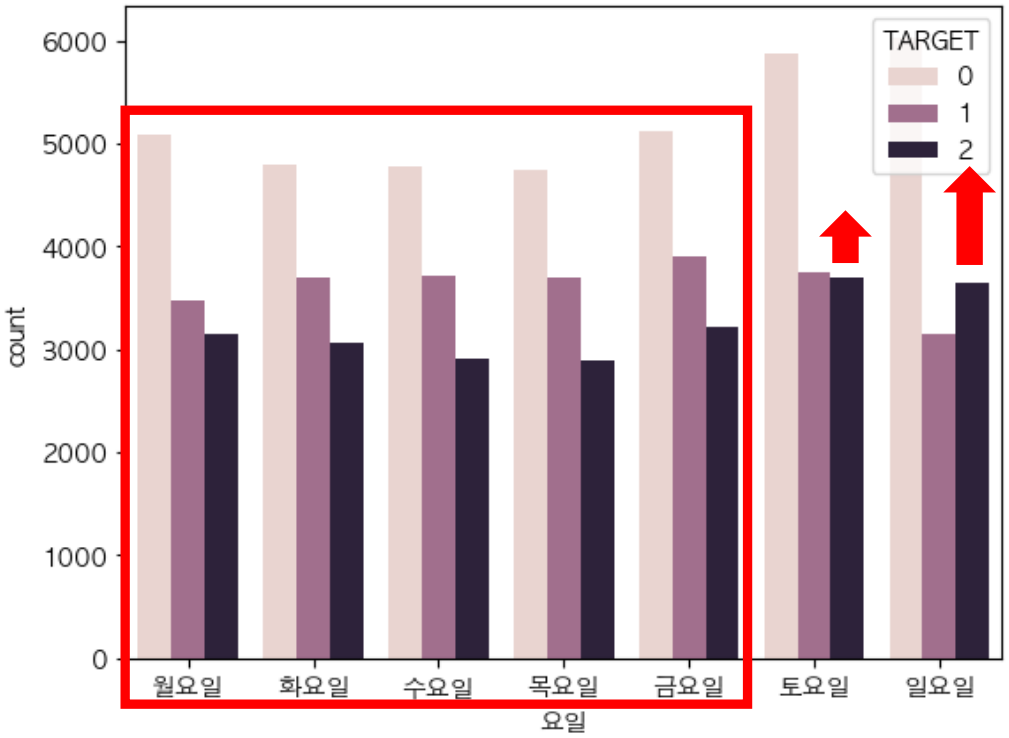
■ PC3: 강수량(-), 안개(-), 번개(-) → 비가 오지 않는 정도

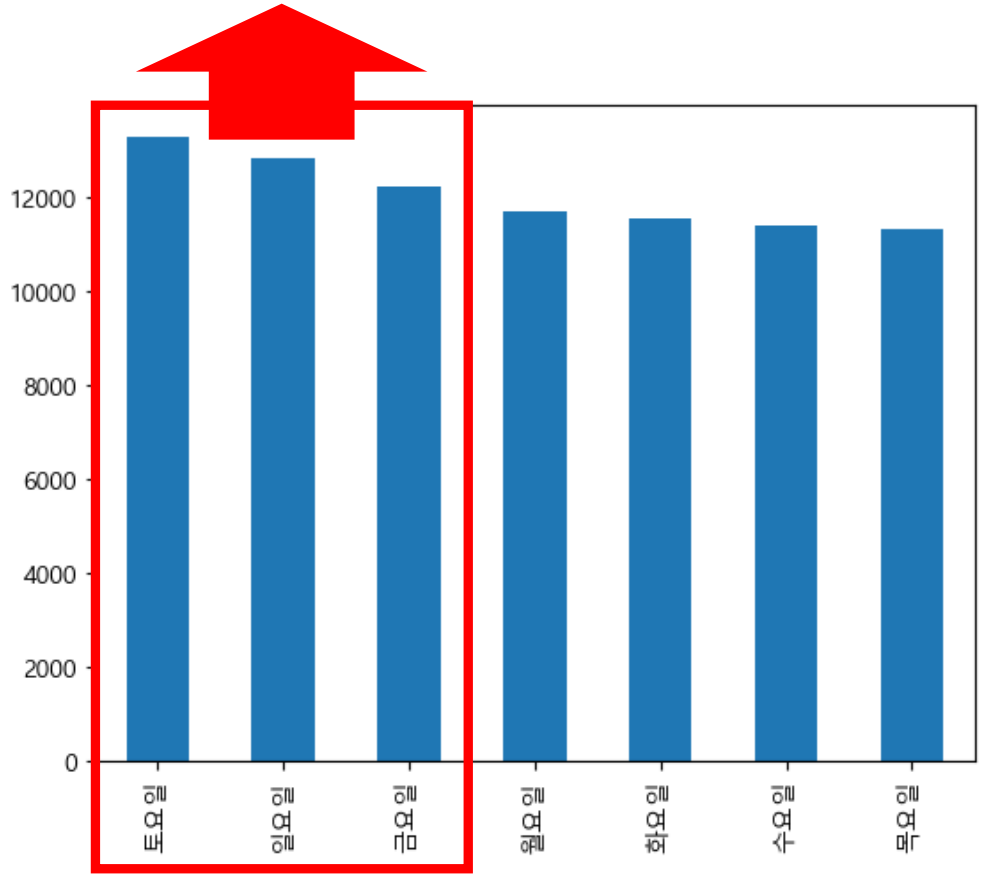
■ PC4: 연기/연무(-) → 시야 확보 정도

■ PC5: 짙은 안개(+) → 시야 방해 정도

데 이 터 전 처 리

가. '요일' 변수 전처리



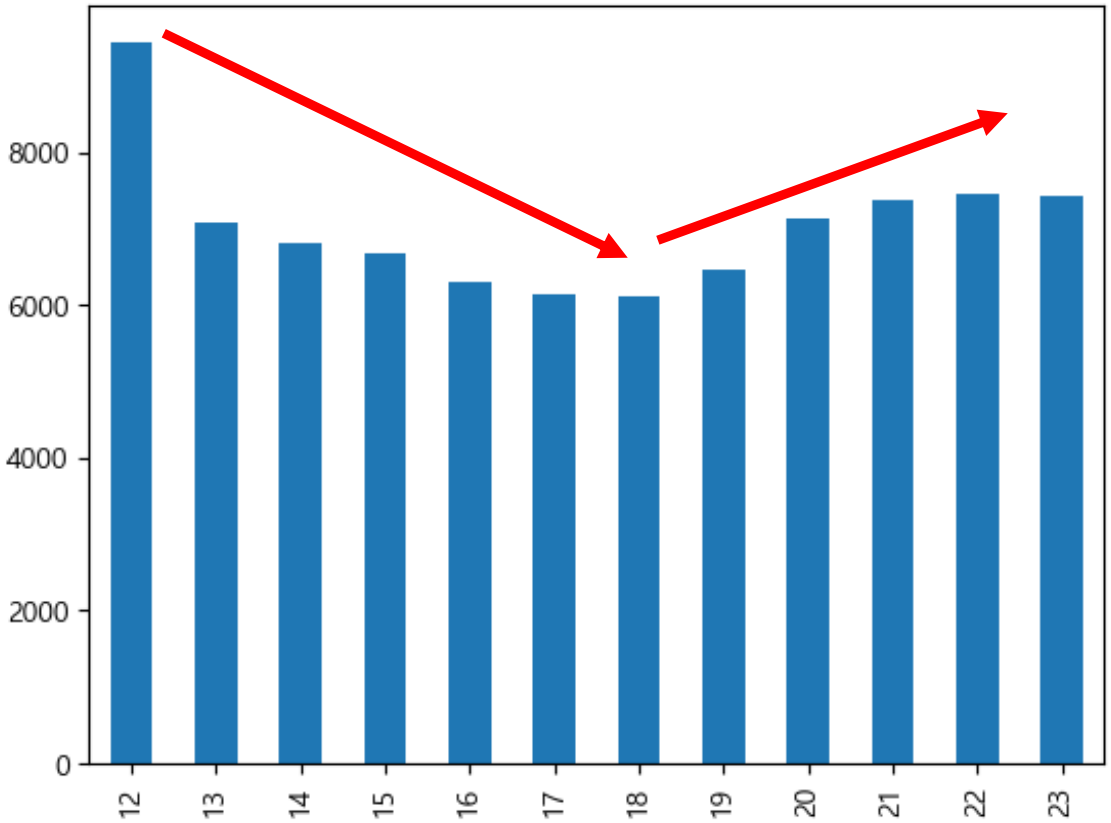


■ 월, 화, 수, 목: 비슷한 분포를 보이며 특이한 경향이 없다. => '평일'이라는 하나의 범주로 고려

■ 금, 토, 일: 특이한 경향을 보인다. (범죄 발생 수가 높음 OR 상해의 비율이 높음) => 각각의 범주로 고려

데 이 터 전 처 리

나. '시간' 변수 전처리



■ 12시: 범죄 발생 수가 높다.

■ 1시~5시: 범죄 발생 수가 감소하는 경향이 있다.

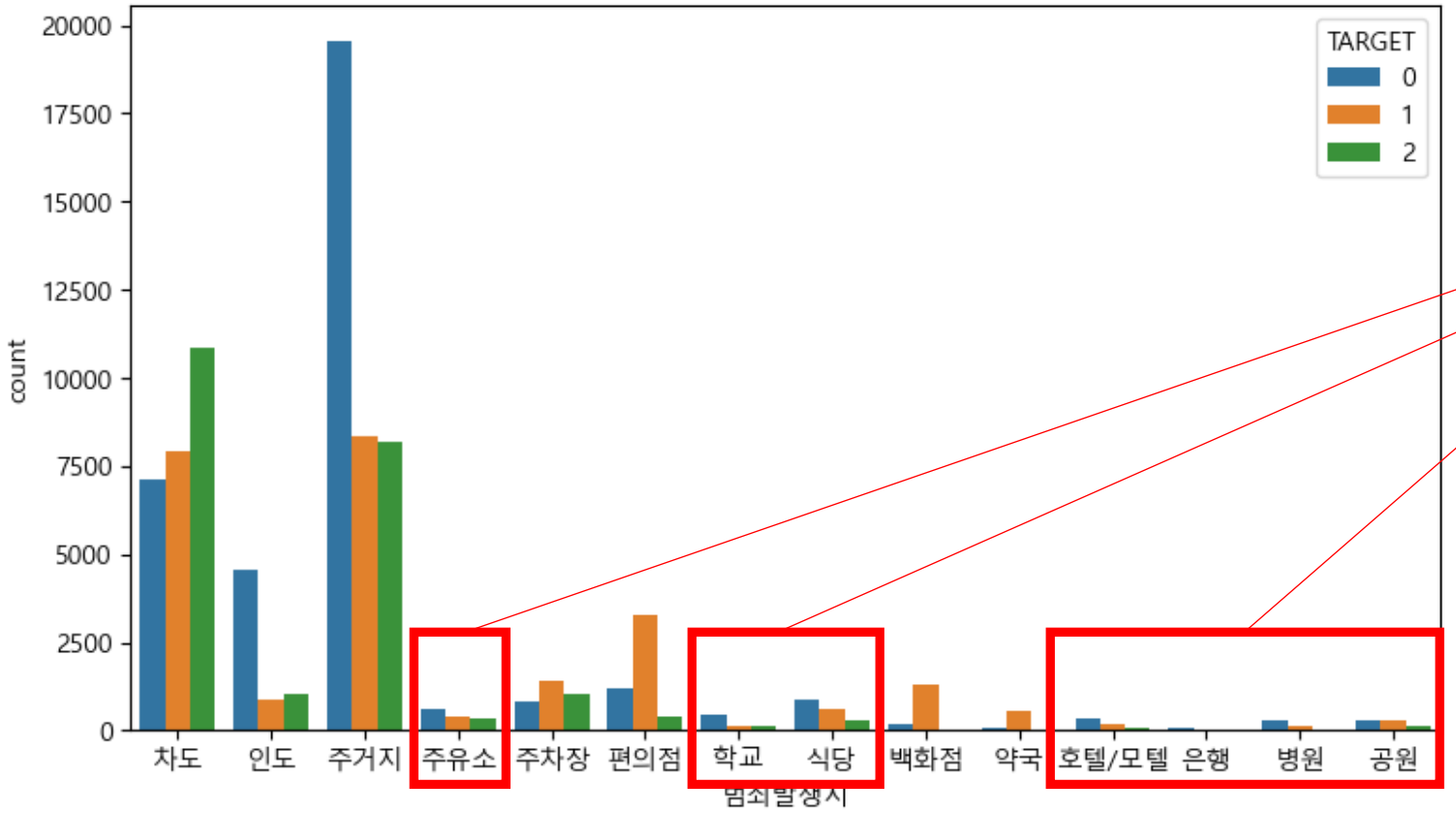
■ 6시 이후: 범죄 발생 수가 증가하는 경향을 보인다.

- 12시는 점심 시간, 6시 이후는 퇴근 시간 이후이므로 유동인구 수의 증가 때문인 것으로 추측

=> 3개의 범주로 재범주화

데 이 터 전 처 리

다. '범죄발생지' 변수 전처리



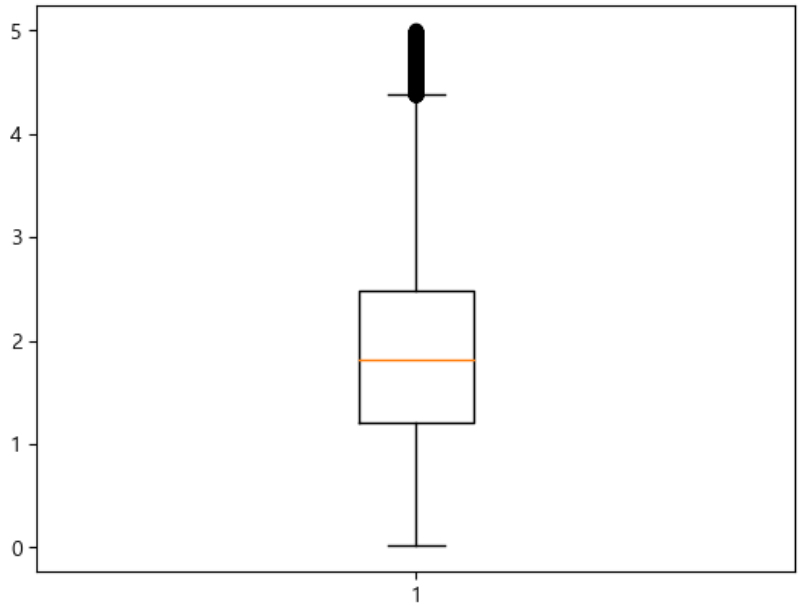
데이터의 수가 매우 적으며,

특이한 경향을 보이지 않는다.

=> 하나의 범주로 고려

데 이 터 전 처 리

라. '사건발생거리' 이상치 처리



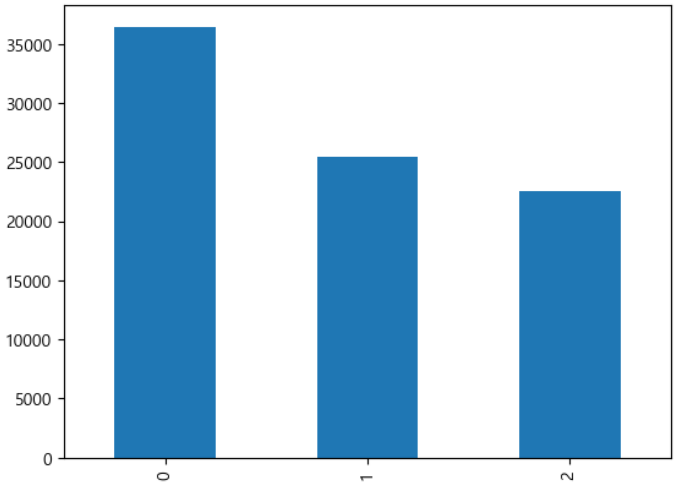
=> 박스플롯 시각화 결과,

이상치가 존재한다는 사실을 알 수 있었다.

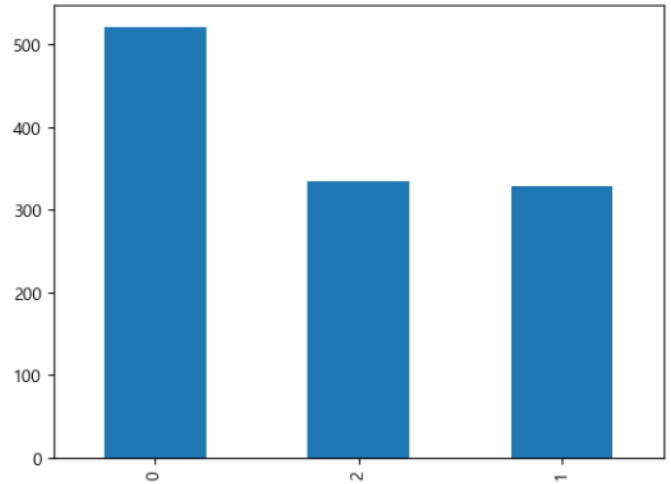
처리?

유지?

<전체 타겟 분포>



<이상치 타겟 분포>



=> 사건 발생 거리가 멀다고 해서 타겟 값에 큰 영향을 주지 않는다고 판단

```
# 이상치를 평균으로 대체
train_df.loc[distance_outlier.index, '사건발생거리'] = np.nan
df_no_outliers = train_df.dropna()
train_df = train_df.fillna(df_no_outliers.mean())
```

=> 따라서 사건 발생 거리 이상치를 평균으로 대체

마. 기상 정보 처리

강수량	강설량	적설량	눈날림	번개
안개	짙은안개	연기/연무	진눈깨비	서리

<5개의 변수>

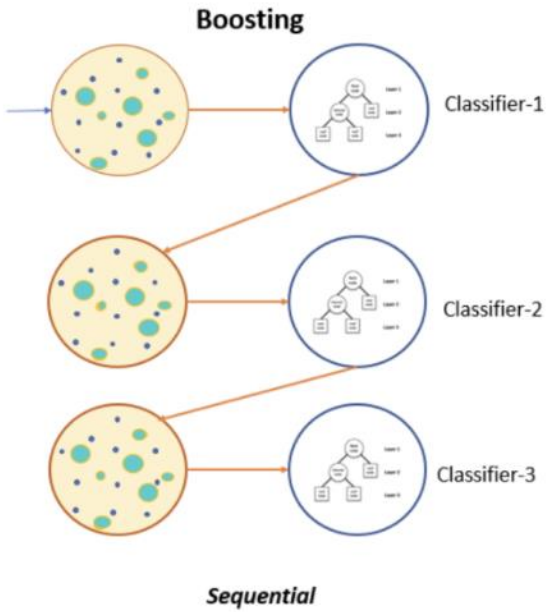
차원 축소

<10개의 변수>

주성분	Loading
PC1	강설량(+), 적설량(+), 눈날림(+), 안개(+)
PC2	진눈깨비(+), 서리(+)
PC3	강수량(–), 안개(–), 번개(–)
PC4	연기/연무(–)
PC5	짙은안개(+)

- 눈이 오는 정도
- 비가 오지 않는 정도
- 시야 확보 정도
- 시야 방해 정도

모 델 링



Boosting 알고리즘

=> 여러 개의 알고리즘이 순차적으로 학습-예측

=> 이전의 알고리즘이 틀린 데이터를 올바르게 예측할 수 있도록 다음 알고리즘에 가중치를 부여

- **CatBoost**
- **XGBoost**

OPTUNA

=> optuna는 하이퍼파라미터 최적화를 도와주는 파이썬의 프레임워크이다.
파라미터의 범위, 목록을 설정하면 매 Trial마다 파라미터를 변경하면서
최적의 파라미터를 찾아낸다.

모 델 링

■ CatBoost

```
# 모델 학습 + optuna

def objectiveCAT(trial: Trial, x_tr, y_tr, x_val, y_val, weight):
    param = {
        'iterations': trial.suggest_int('iterations', 100, 1000),
        'depth': trial.suggest_int('depth', 3, 8),
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.1),
        'random_state': 42,
        'class_weights': weight,
        'cat_features': cat_cols}

    # 학습 모델 생성
    model = CatBoostClassifier(**param)
    cat_model = model.fit(x_tr, y_tr, verbose=True)

    # 모델 성능 확인
    pred = cat_model.predict(x_val)
    score = f1_score(y_val, pred, average="macro")
    return score
```

```
# 하이퍼 파라미터 튜닝

study = optuna.create_study(direction='maximize', sampler=TPESampler(seed=42))
study.optimize(lambda trial: objectiveCAT(trial, x_tr, y_tr, x_val, y_val, weight), n_trials=30)
```

=> **iterations** (반복 횟수): 모델이 학습 데이터를 몇 번 반복해서 학습할지를 결정하는 파라미터

=> **depth** (트리 깊이): 트리가 얼마나 복잡한 패턴을 학습할 수 있는지를 결정

=> **learning_rate** (학습률): 각 시도에서 모델의 가중치를 얼마나 조정할지 결정하는 파라미터

=> **random_state** (랜덤시드): 모델 학습의 재현성을 제공

=> **class_weights** (클래스 가중치): 각 클래스에 대한 손실 함수의 가중치를 설정

=> **cat_features** (범주형 특성): 범주형 변수로 처리해야 하는 열의 인덱스

모 델 링

■ XGBoost

```
# 모델 학습 + optuna

def objectiveXGB(trial: Trial, x_tr, y_tr, x_val, y_val, weight):
    param = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 200),
        'max_depth': trial.suggest_int('max_depth', 5, 20),
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.1),
        'random_state': 42,
        'scale_pos_weight': weight}

    # 학습 모델 생성
    model = XGBClassifier(**param)
    xgb_model = model.fit(x_tr, y_tr, verbose=True)

    # 모델 성능 확인
    pred = xgb_model.predict(x_val)
    score = f1_score(y_val, pred, average="macro")
    return score
```

```
# 하이퍼 파라미터 튜닝

study = optuna.create_study(direction='maximize', sampler=TPESampler(seed=42))
study.optimize(lambda trial: objectiveXGB(trial, x_tr, y_tr, x_val, y_val, weight), n_trials=30)
```

=> **n_estimators** (트리 개수): 많은 트리를 사용할수록 모델의 성능이 향상될 수 있으나,
과적합의 위험이 있으므로 적절한 값을 찾아야 한다.

=> **max_depth** (트리 최대 깊이): 복잡한 패턴을 학습할 수 있는 정도

=> **learning_rate** (학습률): 각 트리의 예측 값에 대한 가중치를 결정하는 파라미터

=> **random_state** (랜덤시드): 모델 학습의 재현성을 제공

=> **scale_pos_weight** (클래스 가중치): 각 클래스에 대한 가중치를 설정

평 가 산 식 산 출

$$\text{F1 Score} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

F1-score: 정밀도(Precision)와 재현율(Recall)의 조화 평균을 나타내는 평가지표
=> macro 평균 이용: 클래스별 F1-score를 산술평균

=> train data를 8(학습용):2(검증용)로 나누어 평가산식 산출

■ CatBoost

```
predictions1 = cat.predict(x_val)

f1 = f1_score(y_val, predictions1, average='macro')
print(f'F1 Score: {f1}')
```

```
F1 Score: 0.5336710133993811
```

>

■ XGBoost

```
predictions2 = xgb.predict(x_val)

f1 = f1_score(y_val, predictions2, average='macro')
print(f'F1 Score: {f1}')
```

```
F1 Score: 0.5253635966198621
```

고 찰

■ 의의

=> 범죄 유형 분류를 통한 범죄 수사 강화 및 예방

=> CatBoost 모델의 우수성: 범주형 변수

■ 아쉬웠던 점

=> 창의적으로 도메인 특성에 맞는 데이터를 구축/수집하는 과정이 없었던 점

=> 타겟 변수 불균형을 해결하기 위한 SMOTE, TomekLink와 같은 기법의 부재

=> binary한 특성을 갖는 데이터에 PCA를 적용한 점