기상청

[Python을 활용한 분석교육실습]

AWS(방재기상관측) 자료를 활용한 고속도로 사고위험 분석사례

기상기후
빅데이터 분석 플랫폼

분석 개요

분석 교육 실습 주제인 AWS(방재기상관측)를 사용한 날씨 정보를 이용하여 날씨가 고속도로 교통사고에 미치는 영향에 대해 알아봅니다.

● 방재기상관측(AWS)[1)]

- 기상청은 서울기상관측소를 비롯하여 전국 95개소의 종관기상관측장비(ASOS)와 무인으로 운영되는 493개소의 자동기상관측장비(AWS)를 이용하여 지상기상관측업무를 수행하고 있습니다.
- 방재기상관측이란 지진 · 태풍 · 홍수 · 가뭄 등 기상현상에 따른 자연재해를 막기 위해 실시하는 지상관측을 말합니다.
- 관측 공백 해소 및 국지적인 기상 현상을 파악하기 위하여 전국 약 510여 지점에 자동기상관측장비(AWS)를 설치하여 자동으로 관측합니다.

● 교통사고[2)]

- 도로교통법 제2조의 규정에 의한 도로(도로법에 의한 도로, 유료도로법에 의한 유료도로, 그 밖의 불특정 다수의 통행을 위하여 공개된 장소)에서 차량의 운행 중 인적인 피해가 발생한 사고를 말한다.
- 고속도로 교통사고 위험도 산출을 위해 아래와 같은 비기상 데이터를 수집합니다.
    - ① 교통류 데이터(시간단위) : 한국도로공사의 교통류 수집기 VDS(Vehicle Detection System)으로부터 수집된 시간단위 데이터
    - ② 도로기하구조 데이터 : 한국도로공사 직접 관리하는 25개 노선에 대한 도로의 평면선형, 종단경사 정보
    - ③ 교통류 데이터(분단위) : VDS의 분단위 데이터는 속도분산 임계치를 구하기 위해 사용되었으며, 사고다발구간 약 250일의 분단위 데이터

1) 출처:기상청 홈페이지

2) 출처:TAAS 교통사고분석시스템

분석 시나리오

실습할 예제는 AWS(방재기상관측) 기상 데이터와 비기상 데이터를 활용하여 날씨가 고속도로에 미치는 영향에 대한 모형을 구축해보는 시간을 가져 봅니다.

● 고속도로 사고 위험도 분석 모형 구축시나리오

[분석 활용 데이터] **[Analytics]**

기상 데이터

- 기온, 풍속 강수량

+

비기상 데이터

- 교통류 / 속도
- 도로기하구조

교통사고 영향 변수 선택	
정의	고속도로의 각 1KM구간 번호 및 콘존에 대한 기상(기온, 풍속, 강수의 평균, 최대, 최소, 편차)과 기하구조(종단, 평면, 경사, 방영) 그리고 각 자동차의 교통량 및 평균 속도
분석방법	Gradient Boosting Model(GBM) & 상관분석
분석결과	교통사고 발생

↓

교통사고 예측 모형 구축	
정의	머신러닝 기법(GBM)을 활용하여 날씨에 따른 교통사고를 예측하는 모형 구축
분석방법	Gradient Boosting Model(GBM)
분석결과	날씨에 따른 교통사고 분석 모형

분석 절차

실습은 데이터 로딩, 데이터 탐색, 데이터 처리, 모형 구축, 모형 검증의 단계에 따라 진행됩니다.

● 사고위험 예측 모형 구축절차

	[실습 설명]	[실습 단계]
1 데이터 로딩	분석 환경을 설정하고 분석에 필요한 기상 데이터 및 비기상 데이터를 로딩하여 분석에 필요한 데이터를 준비하는단계	1. 분석 환경 설정 및 패키지로딩 2. 데이터 불러오기 3. 데이터 결합하기
2 데이터 탐색	분석 데이터의 요약 통계를 확인하는 단계	1. 요약 통계 보기 2. 탐색적 데이터 분석
3 데이터 처리	이상치를 처리하고 최종 데이터 셋을 구성하는 단계	1. 이상치 처리
4 모형 구축	분석할 데이터를 선정하고, 날씨에 따른 고속도로 사고 모형을 구축하는 단계	1. 분석 데이터 셋 2. 모형 구축
5 모형 검증	최종 구축한 산출 모형의성능을 검증하는 단계	1. 변수 중요도 파악 2. 최종 모형 선택 3. 모형 성능 및 예측력 파악

※ 본 실습 교재는 교육용으로 기개발된 사고위험 예측 알고리즘을 최대한 단순화한 모형입니다. 기존 개발된 모형의 데이터와 과정이 일부 달라, 개발된 모형의 결과와는 다를 수 있습니다.

분석 데이터

고속도로 교통사고 위험도 전망 모형 구축에 사용된 파일 및 변수 정보를 확인합니다.

● 고속도로 교통사고 위험도 전망 모형 구축에 사용된 파일 및 변수정보

파일명	파일설명	변수명	변수설명	형식	예제
WEATHER	방제기상 관측(AWS)	TA_MAX_2_STD	이전 2시간 동안 최대 기온의 편차	Num	0.075
		WS_MAX_6_STD	이전 6시간 동안 최대 풍속의 편차	Num	0.281
		WS_MAX_2_MIN	이전 2시간 동안 최대 풍속의 최소	Num	2.45
		TA_MIN_6_MAX	이전 6시간 동안 최소 기온의 최대	Num	4.9
		WS_AVG_2_STD	이전 2시간 동안 평균 풍속의 편차	Num	0.1
		TA_MIN_6_STD	이전 6시간 동안 최소 기온의 편차	Num	0.476
		RN_3_MAX	이전 3시간 동안 최대 강수량	Num	0.125
TRAF	고속도로 교통량	AVG_SPEED	1시간 평균 속도	Num	85
		VOLUME_ALL	1시간 총 교통량	Num	2243
ROAD	고속도로 기하구조	GISID	1KM 구간 번호	Chr	00100000000000597
		CONZONE	콘존 ID	Chr	0010CZE010
		BUSlane	버스전용차로 유무	Num	0
		JD_3KM_STD	이전 3KM 구간 동안 종단 편차	Num	0.816
		PM_3KM_STD	이전 3KM 구간 동안 평면 편차	Num	306
		JD_1KM_AVG	이전 1KM 구간 동안 종단 평균	Num	-3.63
		KS_3KM_AVG	이전 3KM 구간 동안 경사 평균	Num	2.66
		BY_3KM_STD	이전 3KM 구간 동안 방영 편차	Num	0.91
		BY_3KM_AVG	이전 3KM 구간 동안 방영 평균	Num	2.66
		SPEEDLIMIT	제한속도	Num	100
		Month	사고 발생 월	Int	11
		hour	사고 발생 시간	Int	01

기상청

I. 데이터 로딩

1. 분석 환경 설정 및 패키지 로딩
2. 데이터 불러오기 :
    1. 기상 데이터
    2. 도로기하구조 데이터
    3. 교통류 데이터
3. 데이터 결합하기

1. 분석 환경 설정 및 패키지 로딩 (1/2)

- **교육실습 Python 소스**
  - 분석을 실행하기 위한 예제 소스 위치

  ./accident/weather_traf.ipynb

- **패키지 설치 및 로딩**
  - 분석을 위해 사용할 패키지를 설치하고 주피터 노트북에서 로딩

```
#=============================================================
# 패키지 로딩
#=============================================================
import pandas as pd                 # 데이터 분석을 위한 라이브러리
import matplotlib.pyplot as plt     # 시각화를 위한 라이브러리
import seaborn as sns               # 시각화를 위한 라이브러리
import numpy as np                  # 다차원 배열을 위한 라이브러리
import sys                          # 파이썬 인터프리터를 제어할 수 있는 라이브러리

from sklearn.model_selection import train_test_split
                                    # 학습 및 테스트 데이터들을 무작위로 분할
from sklearn.ensemble import GradientBoostingRegressor # 모델을 위한 라이브러리
from sklearn.preprocessing import LabelEncoder # 원-핫 인코딩을 위한 라이브러리
```

1. 분석 환경 설정 및 패키지 로딩 (2/2)

- **분석 환경 설정**
  - **분석을 실행하기 전 메모리를 초기화하고 옵션들을 지정**

```
#=============================================================
#  분석 환경 설정
#=============================================================
# Python 메모리에 생성된 모든 객체 삭제(초기화)
sys.stdout.flush()

# 경고 메세지를 출력되지 않도록 합니다.
import warnings
warnings.filterwarnings(action="ignore")

#=============================================================
# 작업 디렉터리 경로 확인
#=============================================================
import os

currentPath = os.getcwd()  # 현재 위치한 디렉터리 경로 확인
print('Current working dir : %s' % currentPath)
```

2. 데이터 불러오기 : 기상 데이터

- **데이터를 불러온 뒤, 구조 확인하기**

```
# 데이터 로딩
#============================================================
# 데이터 읽어오기
#============================================================
# 기상 데이터
data_weather = pd.read_csv("weather.csv")

# 불러온 데이터 구조 확인하기
data_weather.info()
```

- **pd.read_csv()로 기상 데이터 불러오기**
- **info()로 읽어 온 데이터 구조 확인**

> 실행 결과

```
<class "pandas.core.frame.DataFrame">
RangeIndex: 220064 entries, 0 220063
Data columns (total 11 columns):
#       Column          Non-Null Count      Dtype
---     ------          --------------      -----
0       CONZONE         220064 non-null     object
1       GISID           220064 non-null     int64
2       month           220064 non-null     int64
3       hour            220064 non-null     int64
4       TA_MAX_2_STD    220064 non-null     float64
5       WS_MAX_6_STD    220064 non-null     float64
6       WS_MAX_2_MIN    220064 non-null     float64
7       TA_MIN_6_MAX    220064 non-null     float64
8       WS_AVG_2_STD    220064 non-null     float64
9       TA_MIN_6_STD    220064 non-null     float64
10      RN_3_MAX        220064 non-null     float64
dtypes: float64(7), int64(3), object(1)
memory usage: 18.5+ MB
```

2. 데이터 불러오기 : 도로기하 데이터

- **데이터를 불러온 뒤, 구조 확인하기**

```
#============================================================
# 도로기하 데이터
#============================================================
data_road = pd.read_csv("road.csv")

# 불러온 데이터 구조 확인하기
data_road.info()
```

- **pd.read_csv()로 도로기하 데이터 불러오기**
- **info()로 읽어 온 데이터 구조 확인**

> 실행 결과

```
<class "pandas.core.frame.DataFrame">
RangeIndex: 220064 entries, 0 220063
Data columns (total 13 columns):
#       Column          Non-Null Count      Dtype
---     ------          --------------      -----
0       month           220064 non-null     int64
1       hour            220064 non-null     int64
2       Y               220064 non-null     int64
3       GISID           220064 non-null     int64
4       CONZONE         220064 non-null     object
5       BUSlane         220064 non-null     int64
6       JD_3KM_STD      220064 non-null     float64
7       PM_3KM_STD      220064 non-null     float64
8       JD_1KM_AVG      220064 non-null     float64
9       KS_3KM_AVG      220064 non-null     float64
10      BY_3KM_STD      220064 non-null     float64
11      BY_3KM_AVG      220064 non-null     float64
12      SPEEDLIMIT      220064 non-null     int64
dtypes: float64(6), int64(6), object(1)
memory usage: 21.8+ MB
```

2. 데이터 불러오기 : 교통류 데이터

- **데이터를 불러온 뒤, 구조 확인하기**

```
#=============================================================
# 교통류 데이터
#=============================================================
data_traf = pd.read_csv("traf.csv")

# 불러온 데이터 구조 확인하기
data_traf.info()
```

- **pd.read_csv()로 교통류 데이터 불러오기**
- **info()로 읽어 온 데이터 구조 확인**

> 실행 결과

```
<class "pandas.core.frame.DataFrame">
RangeIndex: 220064 entries, 0 220063
Data columns (total 6 columns):
#       Column          Non-Null Count      Dtype
---     ------          --------------      -----
0       CONZONE         220064 non-null     object
1       GISID           220064 non-null     int64
2       month           220064 non-null     int64
3       hour            220064 non-null     int64
4       AVG_SPEED       220064 non-null     float64
5       VOLUME_ALL      220064 non-null     float64
dtypes: float64(2), int64(3), object(1)
memory usage: 10.1+ MB
```

3. 데이터 결합하기

- **데이터 결합하기**
  - 날짜, 고속도로의 콘존 ID와 1KM의 구간을 기준으로 기상 데이터, 교통류, 도로기하 데이터를 결합

```
#============================================================
# 테이블 결합 및 확인
#============================================================
df_raw = pd.merge(data_weather, data_traf)
weather_traf = pd.merge(df_raw, data_road)
weather_traf.head()
```

  - **pd.merge()로 기상 데이터, 교통류, 도로기하 데이터를 결합**

> 실행 결과

	CONZONE	GISID	month	hour	TA_MAX_2_STD	WS_MAX_6_STD	WS_MAX_2_MIN	TA_MIN_6_MAX	WS_AVG_2_STD
0	0010CZE010	100000000000597	1	0	0.075000	0.280511	2.450000	4.900	0.100000
1	0010CZE010	100000597001000	1	0	0.287500	0.288401	1.975000	4.225	0.262500
2	0010CZE010	100001000002000	1	0	0.300000	0.628466	3.533333	2.700	0.133333
3	0010CZE011	100002000003000	1	0	0.216667	0.613659	2.666667	6.100	0.483333
4	0010CZE011	100003000004000	1	0	0.212500	0.394803	2.275000	3.900	0.225000

5 rows × 22 columns

TA_MIN_6_STD	...	VOLUME_ALL	Y	BUSlane	JD_3KM_STD	PM_3KM_STD	JD_1KM_AVG	KS_3KM_AVG	BY_3KM_STD	BY_3KM_AVG	SPEEDLIMIT
0.476313	...	2243.000000	2	0	0.000000	0.000000	-2.4438	2.000000	0.000000	1.000000	100
1.082753	...	2190.500000	4	0	0.815550	0.000000	-0.8127	1.500000	0.000000	1.000000	100
0.573760	...	1952.666667	3	0	1.305301	306.412939	-4.0098	2.000000	1.414214	2.000000	100
0.750800	...	1993.000000	3	0	3.524292	502.416383	4.5333	2.333333	1.699673	3.333333	100
0.920279	...	1782.000000	4	0	3.507740	351.125176	-0.5324	2.333333	0.816497	4.000000	100

기상청

II. 데이터 탐색

1. 타입 변환

2. 탐색적 자료 분석

1. 타입 변환

- **데이터 요약**
  - R과는 달리 Python에서는 데이터 분석에 사용할 컬럼의 타입이 수치형이기 때문에 타입 변환이 필요하지 않습니다.

```
#============================================================
# 데이터 요약 : 타입확인
#============================================================
weather_traf.info()
```

- **info()를 통해 데이터의 정보 확인**

> 실행 결과

```
<class "pandas.core.frame.DataFrame">
RangeIndex: 220064 entries, 0 220063
Data columns (total 22 columns):
#       Column          Non-Null Count      Dtype
---     ------          --------------      -----
0       CONZONE         220064 non-null     object
1       GISID           220064 non-null     int64
2       month           220064 non-null     int64
3       hour            220064 non-null     int64
4       TA_MAX_2_STD    220064 non-null     float64
5       WS_MAX_6_STD    220064 non-null     float64
6       WS_MAX_2_MIN    220064 non-null     float64
7       TA_MIN_6_MAX    220064 non-null     float64
8       WS_AVG_2_STD    220064 non-null     float64
9       TA_MIN_6_STD    220064 non-null     float64
10      RN_3_MAX        220064 non-null     float64
11      AVG_SPEED       220064 non-null     float64
12      VOLUME_ALL      220064 non-null     float64
13      Y               220064 non-null     int64
14      BUSlane         220064 non-null     int64
15      JD_3KM_STD      220064 non-null     float64
16      PM_3KM_STD      220064 non-null     float64
17      JD_1KM_AVG      220064 non-null     float64
18      KS_3KM_AVG      220064 non-null     float64
19      BY_3KM_STD      220064 non-null     float64
20      BY_3KM_AVG      220064 non-null     float64
21      SPEEDLIMIT      220064 non-null     int64
dtypes: float64(15), int64(6), object(1)
memory usage: 38.6+ MB
```

2. 탐색적 데이터 분석 (1/16)

- **데이터 기술 통계**
  - **describe()를 통해 데이터의 기술 통계 확인**

```
#=============================================================
# 데이터 기술·통계
#=============================================================
weather_traf.describe(include="all")
```

- **describe()를 통해 데이터의 기술 통계 확인**
  - count : 컬럼별 총 데이터수
  - unique : 컬럼별 중복되지 않는 값(유일값)
  - top : 가장 많은 데이터의 수(최빈값인 항목의 개수)
  - freq : 최빈값인 항목의 실제 갯수
  - mean / std : 컬럼별 데이터의 평균 / 표준편차
  - 25% / 50% / 75% : 사분위수의 각 지점으로, 분포를 반영해 평균을 보완하는 목적으로 사용

2. 탐색적 데이터 분석 (2/16)

> 실행 결과

	CONZONE	GISID	month	hour	TA_MAX_2_STD	WS_MAX_6_STD	WS_MAX_2_MIN	TA_MIN_6_MAX	WS_AVG_2_STD
count	220064	2.200640e+05	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000
unique	100	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	0010CZE355	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	5971	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	1.869739e+14	6.539834	11.493338	0.432451	0.560046	2.013745	14.264592	0.202159
std	NaN	2.648517e+14	3.453141	7.064051	0.355987	0.239764	1.082260	9.995684	0.126492
min	NaN	1.000000e+14	1.000000	0.000000	0.000000	0.000000	0.000000	-18.600000	0.000000
25%	NaN	1.002310e+14	4.000000	5.000000	0.200000	0.389807	1.214286	5.550000	0.114286
50%	NaN	1.010280e+14	7.000000	11.000000	0.333333	0.525824	1.850000	14.925000	0.178571
75%	NaN	1.012680e+14	10.000000	18.000000	0.555556	0.692413	2.650000	23.262500	0.262500
max	NaN	1.000067e+15	12.000000	23.000000	12.700000	4.521522	11.000000	37.000000	1.950000

TA_MIN_6_STD	...	VOLUME_ALL	Y	BUSlane	JD_3KM_STD	PM_3KM_STD	JD_1KM_AVG	KS_3KM_AVG	BY_3KM_STD	BY_3KM_AVG
220064.000000	...	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000	220064.000000
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1.321817	...	1661.321725	4.577777	0.251977	2.020977	759.100687	2.509476	2.611624	0.947620	1.994156
0.796737	...	1357.253864	2.162705	0.434149	1.512072	1615.886542	3.852000	0.951685	0.913621	0.992410
0.000000	...	1.000000	1.000000	0.000000	0.000000	0.000000	-6.450000	1.000000	0.000000	1.000000
0.739088	...	589.500000	3.000000	0.000000	0.615598	141.421356	0.000000	2.000000	0.000000	1.000000
1.125422	...	1248.200000	4.000000	0.000000	2.154319	286.123516	2.400000	2.666666	0.942809	2.000000
1.723672	...	2411.000000	6.000000	1.000000	3.304074	801.387685	6.500000	3.333333	1.885618	2.666666
13.907512	...	8689.000000	19.000000	1.000000	5.421152	11785.113020	6.500000	4.000000	2.357023	5.666666

SPEEDLIMIT
220064.000000
NaN
NaN
NaN
101.227007
3.280940
100.000000
100.000000
100.000000
100.000000
110.000000

2. 탐색적 데이터 분석 (3/16)

● 데이터 결측치 파악

```
#============================================================
# 결측치 파악
#============================================================
weather_traf.isnull().sum()
```

▪ **isnull().sum()를 통해 데이터의 결측치 파악**

> 실행 결과

```
# 결측치 파악

CONZONE          0
GISID            0
month            0
hour             0
TA_MAX_2_STD     0
WS_MAX_6_STD     0
WS_MAX_2_MIN     0
TA_MIN_6_MAX     0
WS_AVG_2_STD     0
TA_MIN_6_STD     0
RN_3_MAX         0
AVG_SPEED        0
VOLUME_ALL       0
Y                0
BUSlane          0
JD_3KM_STD       0
PM_3KM_STD       0
JD_1KM_AVG       0
KS_3KM_AVG       0
BY_3KM_STD       0
BY_3KM_AVG       0
SPEEDLIMIT       0
dtype:   int64
```

2. 탐색적 데이터 분석 (4/16)

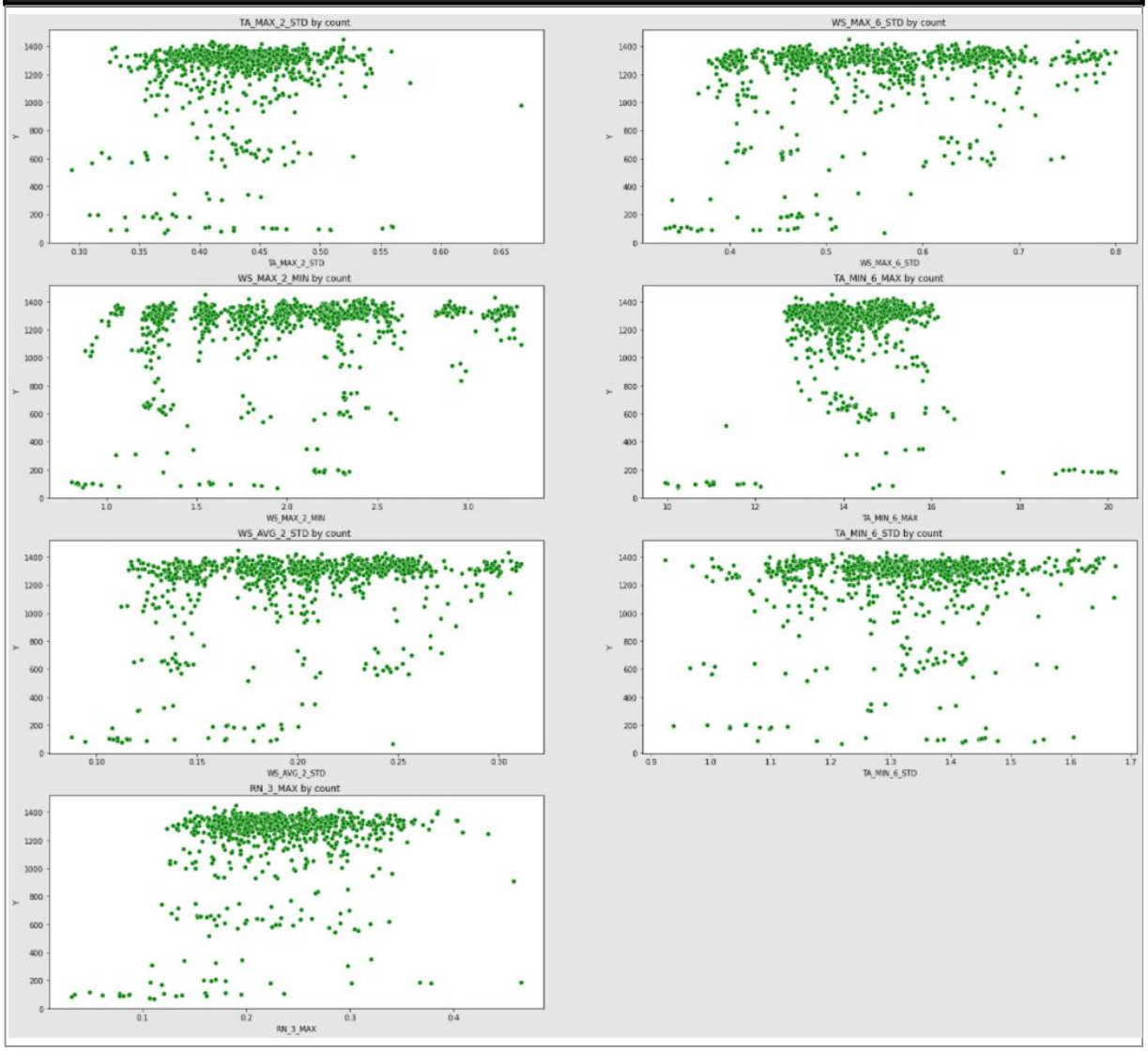
- **1KM 구간에 따른 사고 빈도 (기상)**

```
#============================================================
# 1KM 구간에 따른 사고 빈도 (기상)
#============================================================
plt.figure(figsize=(26, 24))
for i, col in enumerate(weather_traf.filter(regex="TA|WS|RN")):
    weather_traf_1km = weather_traf.groupby("GISID")[[col, "Y"]].agg({col:"mean", "Y":"sum"})
    plt.subplot(4, 2, i + 1)
    sns.scatterplot(data=weather_traf_1km, x=col, y="Y", color="g")
    plt.title(col + " by count ")
```

- plt.figure(figsize = (26, 24))를 통해 figsize가 가로 26, 세로 24인치인 그래프를 그려줄 창을 하나 생성합니다.
- for 반복문을 통해 기상 데이터에 속해있던 1km 구간에 따른 사고 빈도 관련 컬럼 데이터를 i, col 변수에 전달 받아, weather_traf 데이터의 "GISID" 그룹별로 사고 빈도 그래프를 시각화 합니다.
    - weather_traf의 테이블에서 특정 컬럼인 "GISID"를 groupby로 그룹화 시킨 후, "col"은 mean값, "Y"는 sum값을 계산해 줍니다.
    - plt.subplot으로 4행 2열의 그래프를 그려줍니다.
    - data는 weather_traf_1km, x축은 "col", y축은 "Y"인 포인트(점)의 색깔이 "g"(초록색)인 scatterplot을 그려줍니다.
    - plt.title로 그래프의 제목을 붙여줍니다.

2. 탐색적 데이터 분석 (5/16)

> 실행 결과



2. 탐색적 데이터 분석 (6/16)

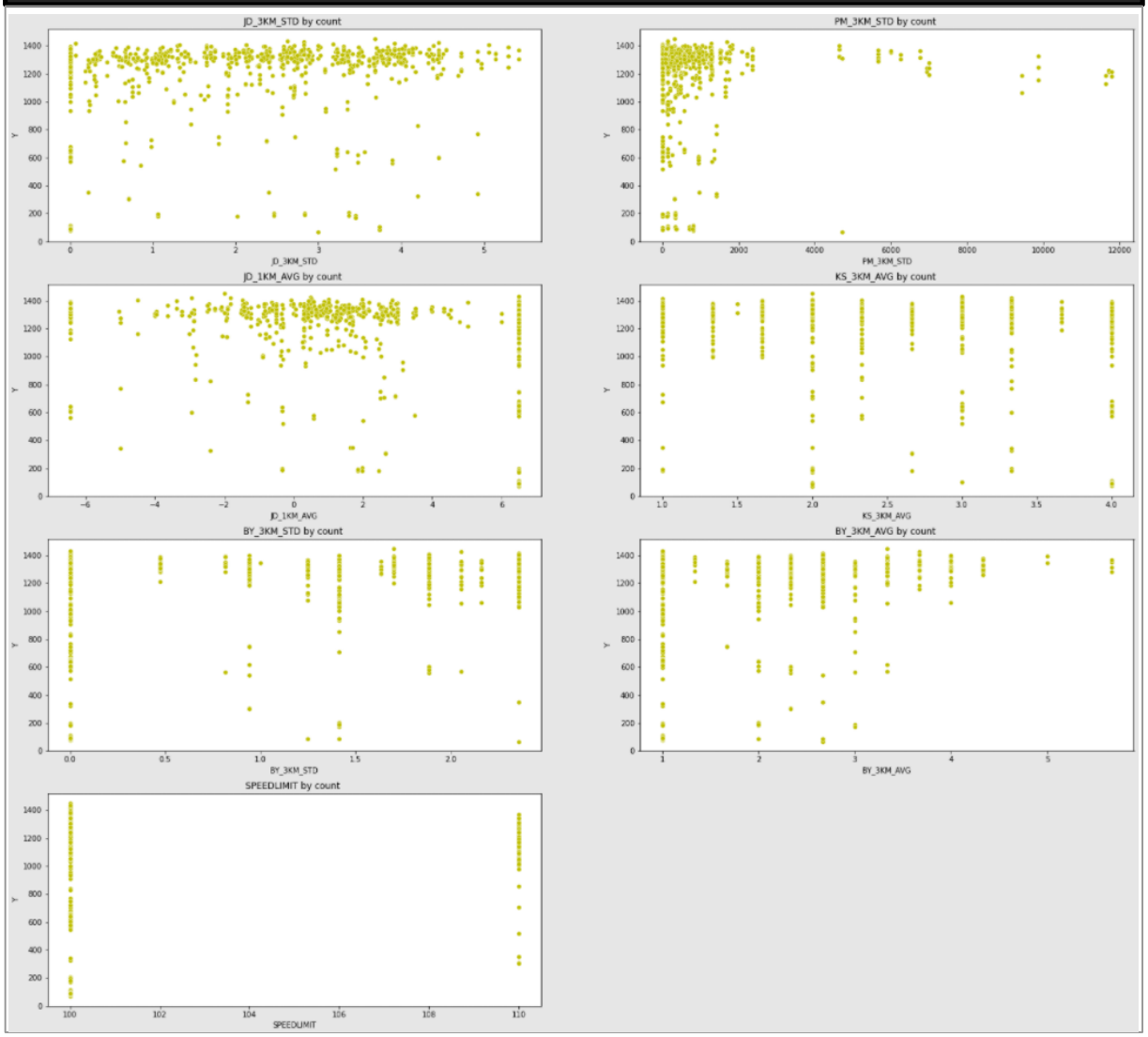
- **1KM 구간에 따른 사고 빈도 (기하구조)**

```
#============================================================
# 1KM 구간에 따른 사고 빈도 (기하구조)
#============================================================
plt.figure(figsize=(26, 24))
for i, col in enumerate(weather_traf.filter(regex="JD|PM|KS|BY|LIMIT")):
    weather_traf_1km = weather_traf.groupby("GISID")[[col, "Y"]].agg({col:"mean", "Y":"sum"})
   plt.subplot(4, 2, I + 1)
    sns.scatterplot(data=weather_traf_1km, x=col, y="Y", color="y")
    plt.title(col + " by count ")
```

- plt.figure(figsize = (26, 24))를 통해 figsize가 가로 26, 세로 24인치인 그래프를 그려줄 창을 하나 생성합니다.
- for 반복문을 통해 기하구조 데이터에 속해있던 1km 구간에 따른 사고 빈도 관련 컬럼 데이터를 i, col 변수에 전달 받아, weather_traf 데이터의 "GISID" 그룹별로 사고 빈도 그래프를 시각화 합니다.
    - weather_traf의 테이블에서 특정 컬럼인 "GISID"를 groupby로 그룹화 시킨 후, "col"은 mean값, "Y"는 sum값을 계산해 줍니다.
    - plt.subplot으로 4행 2열의 그래프를 그려줍니다.
    - data는 weather_traf_1km, x축은 "col", y축은 "Y"인 포인트(점)의 색깔이 "y"(노랑색)인 scatterplot을 그려줍니다.
    - plt.title로 그래프의 제목을 붙여줍니다.

2. 탐색적 데이터 분석 (7/16)

> 실행 결과



2. 탐색적 데이터 분석 (8/16)

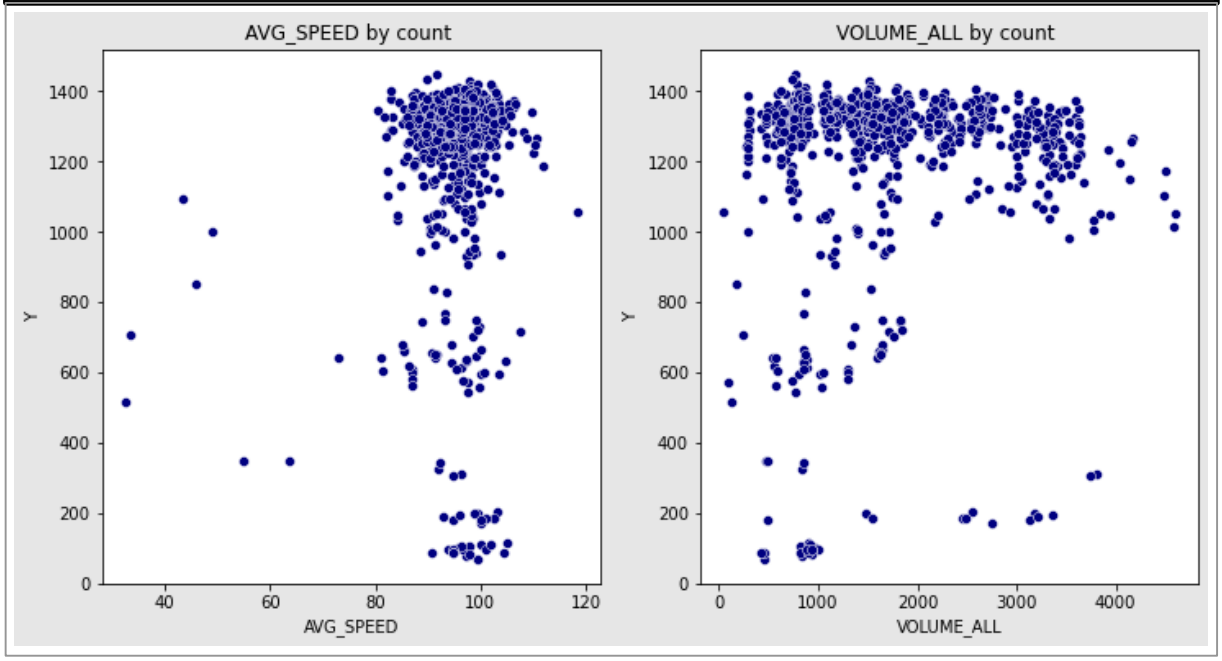
- **1KM 구간에 따른 사고 빈도 (교통류)**

```
#============================================================
# 1KM 구간에 따른 사고 빈도 (교통류)
#============================================================
plt.figure(figsize=(12, 6))
for i, col in enumerate(weather_traf.filter(regex="_SPEED|VOLUME")):
    weather_traf_1km = weather_traf.groupby("GISID")[[col, "Y"]].agg({col:"mean", "Y":"sum"})
    plt.subplot(1, 2, i + 1)
    sns.scatterplot(data=weather_traf_1km, x=col, y="Y", color="navy")
    plt.title(col + " by count ")
```

- plt.figure(figsize = (12, 6))를 통해 figsize가 가로 12, 세로 6인치인 그래프를 그려줄 창을 하나 생성합니다.
- for 반복문을 통해 교통류 데이터에 속해있던 1km 구간에 따른 사고 빈도 관련 컬럼 데이터를 i, col 변수에 전달 받아, weather_traf 데이터의 "GISID" 그룹별로 사고 빈도 그래프를 시각화 합니다.
    - weather_traf의 테이블에서 특정 컬럼인 "GISID"를 groupby로 그룹화 시킨 후, "col"은 mean값, "Y"는 sum값을 계산해 줍니다.
    - plt.subplot으로 1행 2열의 그래프를 그려줍니다.
    - data는 weather_traf_1km, x축은 "col", y축은 "Y"인 포인트(점)의 색깔이 "navy"(남색)인 scatterplot을 그려줍니다.
    - plt.title로 그래프의 제목을 붙여줍니다.

2. 탐색적 데이터 분석 (9/16)

> 실행 결과



2. 탐색적 데이터 분석 (10/16)

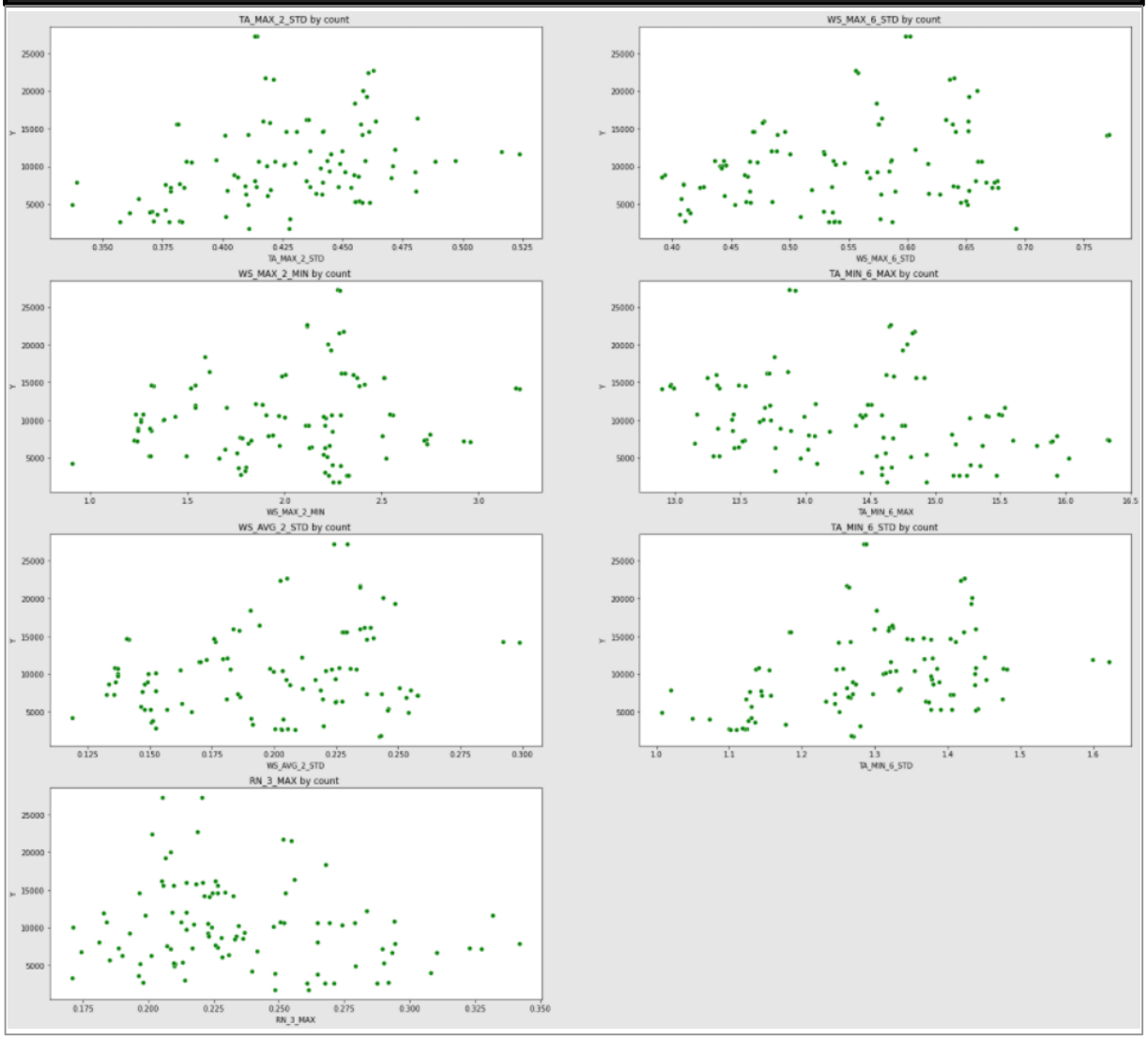
- **콘존에 따른 사고 빈도 (기상)**

```
#============================================================
# 콘존에 따른 사고 빈도 (기상)
#============================================================
plt.figure(figsize=(26, 24))
for i, col in enumerate(weather_traf.filter(regex="TA|WS|RN")):
    weather_traf_con = weather_traf.groupby("CONZONE")[[col, "Y"]].agg({col:"mean", "Y":"sum"})
    plt.subplot(4, 2, i + 1)
    sns.scatterplot(data=weather_traf_con, x=col, y="Y", color="g")
    plt.title(col + " by count ")
```

- plt.figure(figsize = (26, 24))를 통해 figsize가 가로 26, 세로 24인치인 그래프를 그려줄 창을 하나 생성합니다.
- for 반복문을 통해 기상 데이터에 속해있던 콘존에 따른 사고 빈도 관련 컬럼 데이터를 i, col 변수에 전달 받아, weather_traf 데이터의 "CONZONE"그룹별로 사고 빈도 그래프를 시각화 합니다.
    - weather_traf의 테이블에서 특정 컬럼인 "CONZONE"를 groupby로 그룹화 시킨 후, "col"은 mean값, "Y"는 sum값을 계산해 줍니다.
    - plt.subplot으로 4행 2열의 그래프를 그려줍니다.
    - data는 weather_traf_con, x축은 "col", y축은 "Y"인 포인트(점)의 색깔이 "g"(초록색)인 scatterplot을 그려줍니다.
    - plt.title로 그래프의 제목을 붙여줍니다.

2. 탐색적 데이터 분석 (11/16)

> 실행 결과



2. 탐색적 데이터 분석 (12/16)

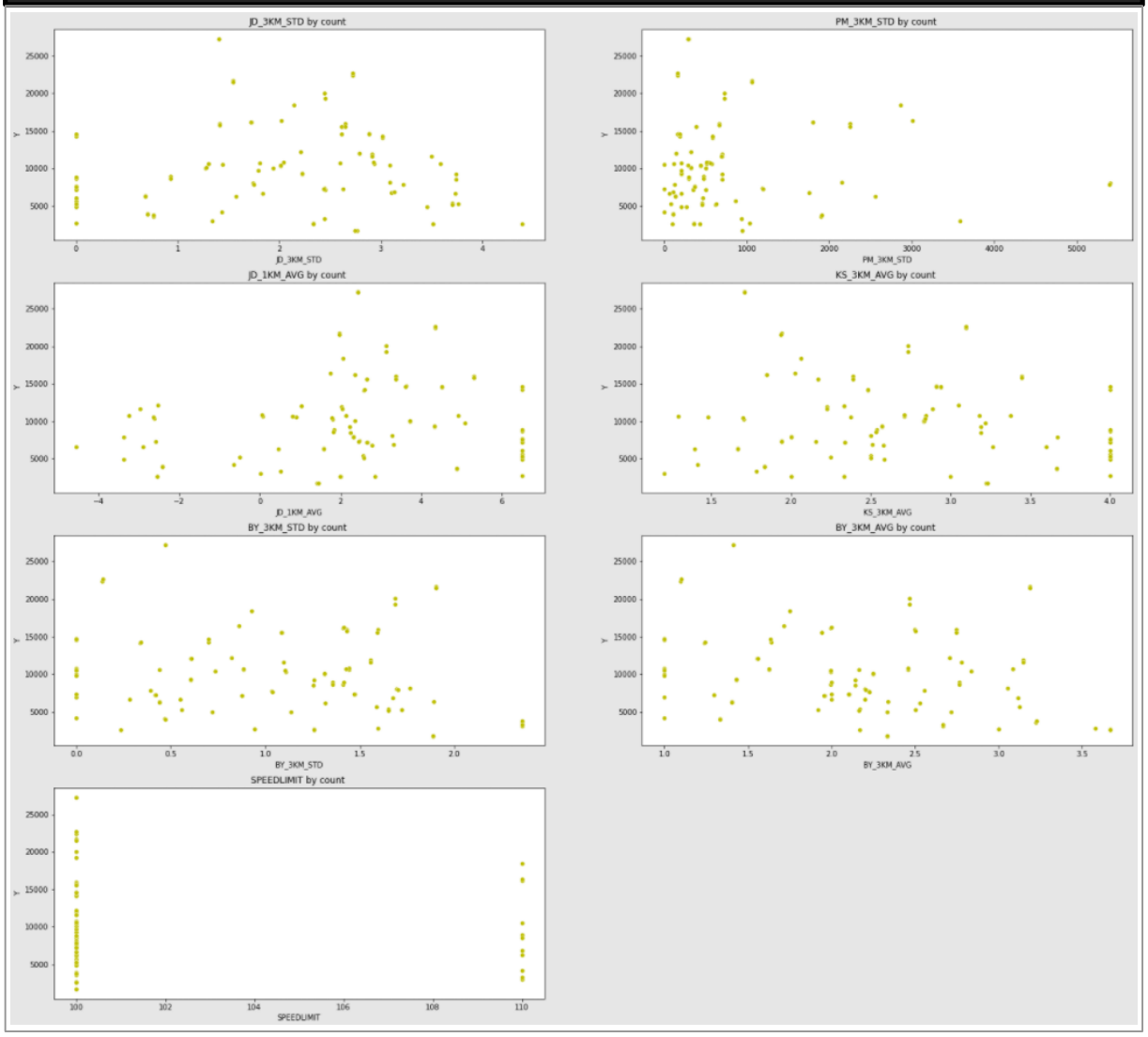
- **콘존에 따른 사고 빈도 (기하구조)**

```
#============================================================
# 콘존에 따른 사고 빈도 (기하구조)
#============================================================
plt.figure(figsize=(26, 24))
for i, col in enumerate(weather_traf.filter(regex="JD|PM|KS|BY|LIMIT")):
    weather_traf_con = weather_traf.groupby("CONZONE")[[col, "Y"]].agg({col:"mean", "Y":"sum"})
    plt.subplot(4, 2, i + 1)
    sns.scatterplot(data=weather_traf_con, x=col, y="Y", color="y")
    plt.title(col + " by count ")
```

- plt.figure(figsize = (26, 24))를 통해 figsize가 가로 26, 세로 24인치인 그래프를 그려줄 창을 하나 생성합니다.
- for 반복문을 통해 기하구조 데이터에 속해있던 콘존에 따른 사고 빈도 관련 컬럼 데이터를 i, col 변수에 전달 받아, weather_traf 데이터의 "CONZONE"그룹별로 사고 빈도 그래프를 시각화 합니다.
    - weather_traf의 테이블에서 특정 컬럼인 "CONZONE"를 groupby로 그룹화 시킨 후, "col"은 mean값, "Y"는 sum값을 계산해 줍니다.
    - plt.subplot으로 4행 2열의 그래프를 그려줍니다.
    - data는 weather_traf_con, x축은 "col", y축은 "Y"인 포인트(점)의 색깔이 "y"(노랑색)인 scatterplot을 그려줍니다.
    - plt.title로 그래프의 제목을 붙여줍니다.

2. 탐색적 데이터 분석 (13/16)

> 실행 결과



2. 탐색적 데이터 분석 (14/16)

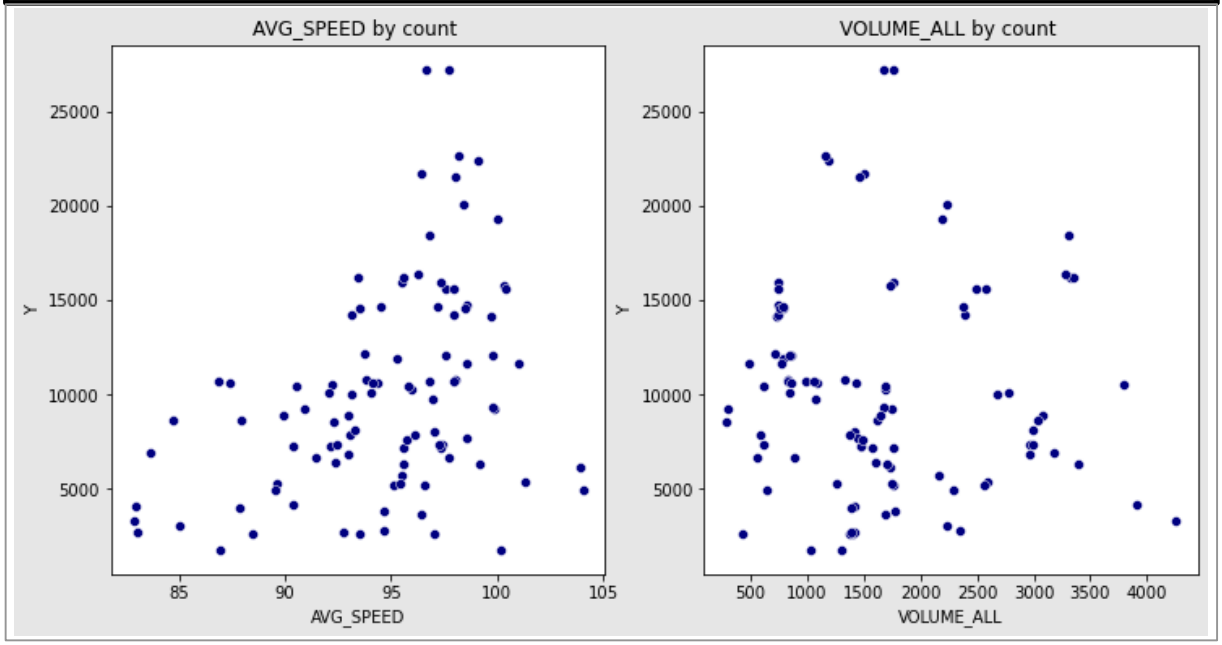
- **콘존에 따른 사고 빈도 (교통류)**

```
#============================================================
# 콘존에 따른 사고 빈도 (교통류)
#============================================================
plt.figure(figsize=(12, 6))
for i, col in enumerate(weather_traf.filter(regex="_SPEED|VOLUME")):
    weather_traf_con = weather_traf.groupby("CONZONE")[[col, "Y"]].agg({col:"mean", "Y":"sum"})
    plt.subplot(1, 2, i + 1)
    sns.scatterplot(data=weather_traf_con, x=col, y="Y", color="navy")
    plt.title(col + " by count ")
```

- plt.figure(figsize = (12, 6))를 통해 figsize가 가로 12, 세로 6인치인 그래프를 그려줄 창을 하나 생성합니다.
- for 반복문을 통해 교통류 데이터에 속해있던 콘존에 따른 사고 빈도 관련 컬럼 데이터를 i, col 변수에 전달 받아, weather_traf 데이터의 "CONZONE"그룹별로 사고 빈도 그래프를 시각화 합니다.
    - weather_traf의 테이블에서 특정 컬럼인 "CONZONE"를 groupby로 그룹화 시킨 후, "col"은 mean값, "Y"는 sum값을 계산해 줍니다.
    - plt.subplot으로 1행 2열의 그래프를 그려줍니다.
    - data는 weather_traf_con, x축은 "col", y축은 "Y"인 포인트(점)의 색깔이 "navy"(남색)인 scatterplot을 그려줍니다.
    - plt.title로 그래프의 제목을 붙여줍니다.

2. 탐색적 데이터 분석 (15/16)

> 실행 결과



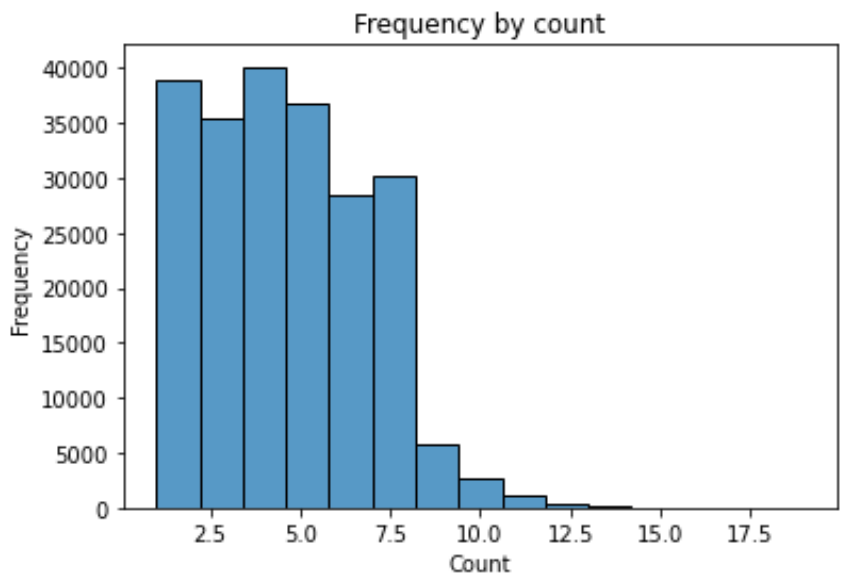
2. 탐색적 데이터 분석 (16/16)

- **히스토그램(발생건수, 빈도)**

```
#============================================================
# 히스토그램 (발생건수, 빈도)
#============================================================
sns.histplot(data=weather_traf, x="Y", bins=15)
plt.title("Frequency by count")
plt.xlabel("Count")
plt.ylabel("Frequency")
```

  - sns.histplot() 함수를 사용해 히스토그램 생성
    - 월, 일, 1km 구간, 콘존 ID에 각각의 발생건수에 대한 빈도 시각화

> 실행 결과



기상청

III. 데이터 처리

1. 이상치 제거

1. 이상치 제거

● **이상치 제거**

- loc을 이용하여 조건문을 통해 이상치 확인 후, 제거 전과 제거 후 행과 열 비교

```
#=============================================================
# 이상치 제거 전
#=============================================================
Print( " 이상치 제거 전: " , weather_traf.shape)

#=============================================================
# 이상치 제거
#=============================================================
weather_traf = weather_traf.loc[(weather_traf["TA_MAX_2_STD"] < 7) &
                                (weather_traf["WS_MAX_2_MIN"] < 7) &
                                (weather_traf["TA_MIN_6_STD"] < 10) &
                                (weather_traf["RN_3_MAX"] < 20) &
                                (weather_traf["PM_3KM_STD"] < 10000)]

#=============================================================
# 이상치 제거 후
#=============================================================
print("이상치 제거 후:", weather_traf.shape)
```

- **이상치 제거 : loc()함수를 통해 이상치로 판단되는 데이터 제거**
  - **TA_MAX_2_STD < 7 : 이전 2시간 동안 최대 기온의 편차(7보다 큰 값 제거)**
  - **WS_MAX_2_MIN < 7 : 이전 2시간 동안 최대 풍속의 최소(7보다 큰 값 제거)**
  - **TA_MIN_6_STD < 10 : 이전 6시간 동안 최소 기온의 편차(10보다 큰 값 제거)**
  - **RN_3_MAX < 20 : 이전 3시간 동안 강수의 최대: 20보다 큰 값 제거**
  - **PM_3KM_STD < 10000 : 이전 3KM 구간 동안 평면 편차(10000보다 큰 값 제거)**
  - **월, 일, 1km 구간, 콘존 ID에 각각의 발생건수에 대한 빈도 시각화**

> 실행 결과

```
이상치 제거 전: (220064, 22)
이상치 제거 후: (218245, 22)
```

기상청

IV. 모형 구축

1. 분석 데이터 셋
2. 모형 구축

1. 분석 데이터 셋

● 범주형 데이터를 수치데이터로 변환

- 기계가 이해할 수 있도록 범주형 데이터를 수치 데이터로 변환

```
#============================================================
# 범주형 데이터를 수치데이터로 변환
#============================================================
le = LabelEncoder()
weather_traf["CONZONE"] = le.fit_transform(weather_traf["CONZONE"])
weather_traf.columns[weather_traf.dtypes.values == "object"]
```

● 데이터 셋 분할

- 모델 학습 및 최적화, 테스트를 위해 데이터 셋 분할

```
#============================================================
# 데이터 셋 분할
#============================================================
# set x and y
X = weather_traf.drop("Y", axis=1)
Y = weather_traf["Y"]

# split dataset
X, X_test, y, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=1234)
X_train.shape, X_test.shape, X_valid.shape
```

- **train_test_split()으로 데이터 셋을 분할**
  1) train과 test 데이터를 각각 8:2 비율로 분할
  2) 검증을 위해 6 : 2 : 2로 train : valid : test를 나누는 작업

> 실행 결과

```
((130947, 21), (43649, 21), (43649, 21))
```

2. 모형 구축(1/5)

- **하이퍼 파라미터 최적화(Hyper-parameter Optimization)**

- GBM 모델의 하이퍼 파라미터를 조정하여 RMSE(Root Mean Square Error)가 가장 높은 모형 선택

```
#============================================================
# 모형 튜닝 자동화
# ============================================================
# cartesian grid search
rate = [0.1, 0.05, 0.001]
depth = [12, 15, 18]
result1 = []
result2 = []
result3 = []

for i in rate: # learning_rate를 0.1, 0.05, 0.001 로 변경하면서 결과 확인
    for j in depth: # max_depth를 12, 15, 18 로 변경하면서 결과 확인
      result1.append(i)
      result2.append(j)
      Traffic_GBM_Model = GradientBoostingRegressor(learning_rate=i, max_depth=j,
                                        subsample=0.85,
                                        min_samples_leaf=0.075,
                                        max_features=0.51,
                                       min_impurity_decrease=0.0,
                                       n_estimators=148,
                                       random_state=1234)
      # 분석환경을 고려한 빠른 결과 산출을 위해, 10% 의 데이터만 사용
      Traffic_GBM_Model.fit(X_train.sample(frac=0.1, random_state=1234), _
                            train.sample(frac=0.1, random_state=1234)) # training_frame = X_train
      pred = Traffic_GBM_Model.predict(X_valid.sample(frac=0.1, random_state=1234))
      # validation_frame = X_valid
      result3.append(np.sqrt(np.sum((y_valid.sample(frac=0.1,
                                                    random_state=1234) - pred) ** 2) / len(pred)))
```

- **For문 안에 For문 배치**

  1) learning rate와 max depth를 변경하면서 결과 확인

  2) train 데이터로 학습하고 valid 데이터로 예측하여 결과 확인

*** Python과 R의 파라미터가 상이하여 모형 결과가 다를 수 있습니다.**

2. 모형 구축(2/5)

- 동일한 조건에서의 R작업물과의 비교를 위해 GradientBoostingRegressor의 변수를 다음과 같이 설정해 줌
  - subsample=0.85, # col_sample_rate
    : 샘플 비율(1보다 작으면 확률적 그라데이션 부스팅 발생)
  - min_samples_leaf=0.075, # min_rows: 218245의 0.075=16384.0
    : 리프 노드(자식이 없는 노드)가 되기 위한 최소한의 샘플 데이터 수
  - max_features=0.51, # col_sample_rate * col_sample_rate_per_tree = 0.51
    : 데이터의 feature를 참조할 비율
  - min_impurity_decrease=0.0, # min_split_improvement
    : 최소 불순도로 이 값 이상으로 감소하면 노드가 분할
  - n_estimators=148, # ntrees
    : 부스팅 단계의 수
  - random_state=1234) # seed
    : 무작위로 seed값을 생성하여 분할을 제어
- frac=0.1 : 분석환경을 고려하여 빠른 결과로 하이퍼 파라미터를 찾기 위해, 10%의 데이터만 사용하지만, 39페이지에선 모든 데이터로 모델을 돌림

* Python과 R의 파라미터가 상이하여 모형 결과가 다를 수 있습니다.

2. 모형 구축(3/5)

```
#============================================================
# RMSE가 낮은 순으로 정렬하기
#============================================================
gbm_gridperf = pd.DataFrame({"learning_rate" : result1,
                             "max_depth" : result2,
                             "RMSE" : result3}).sort_values(by="RMSE")
gbm_gridperf
```

- 위에서 만든 result1, result2, result3을 각각 learning_rate, max_depth, RMSE 컬럼으로 데이터프레임으로 만든 후, RMSE가 낮은 순으로 정렬하기

> 실행 결과

	learning_rate	max_depth	RMSE
0	0.100	12	1.822267
1	0.100	15	1.822267
2	0.100	18	1.822267
3	0.050	12	1.879486
4	0.050	15	1.879486
5	0.050	18	1.879486
6	0.001	12	2.109870
7	0.001	15	2.109870
8	0.001	18	2.109870

*** Python과 R의 파라미터가 상이하여 모형 결과가 다를 수 있습니다.**

2. 모형 구축(4/5)

```
#============================================================
# 모형 선택
#============================================================
# create Model
Traffic_GBM_Model =
GradientBoostingRegressor(max_depth=gbm_gridperf.loc[0][“max_depth”],
                          learning_rate=gbm_gridperf.loc[0][“learning_rate”],
                          subsample=0.85, # col_sample_rate
                          min_samples_leaf=0.075, # min_rows: 218245의 0.075=16384.0
                          max_features=0.51, # col_sample_rate * col_sample_rate_per_tree = 0.51
                          min_impurity_decrease=0.0, # min_split_improvement
                          n_estimators=148, # ntrees
                          random_state=1234) # seed

# Model Summary
Traffic_GBM_Model
Traffic_GBM_Model.fit(X_train, y_train)  # training_frame = train
pred = Traffic_GBM_Model.predict(X_valid)  # validation_frame = valid
```

- GradientBoostingRegressor를 활용하여 모형을 구축한 "Traffic_GBM_Model"를 생성
- **RMSE값 순서로 파라미터를 정렬한 데이터프레임 gbm_gridperf의 RMSE가 가장 낮은 첫번째 행의 파라미터를 찾기 위해 .loc[0]으로 max_depth와 learning_rate를 설정**
  - **gbm_gridperf은 sort_values(by="RMSE")으로 RMSE 값이 낮은순으로 (기본값은 ascending=True로 오름차순) 정렬되어 있으며, 파이썬의 인덱스 순서는 0번째부터 시작하기 때문에 gbm_gridperf.loc[0]로 위치를 찾아 해당 파라미터 값(max_depth, learning_rate)을 불러오기**
- **위의 for문을 이용하여 찾은 값들을 각각 max_depth, learning_rate에 할당한 후, 추가적인 매개변수를 지정**
- **36페이지의** # cartesian grid search**로 찾은 파라미터로 전체 데이터의 성능을 확인하기 위해 학습은 train 전체데이터로, 예측은 valid 전체데이터로 진행**

2. 모형 구축(5/5)

- **GradientBoostingRegressor의 성능을 다양한 평가 지표로 확인**

- GradientBoostingRegressor 모형 성능을 MSE(Mean Squared Error), RMSE(Root Mean Square Error), MAE(Mean Absolute Error), RMSLE(Root Mean Square Logarithmic Error) 평가 지표들로 확인

```
# __RMSE ----
print("MAE :", np.mean(abs(y_valid - pred)) )
print("MSE :", np.sum((y_valid - pred) ** 2) / len(pred) )
print("RMSE :", np.sqrt(np.sum((y_valid - pred) ** 2) / len(pred)) )
print("RMSLE :", np.sqrt(np.mean(np.power(np.log1p(pred) - np.log1p(y_valid), 2))))
```

- **y_valid와 pred로 MAE, MSE, RMSE, RMSLE를 계산**

> 실행 결과

```
MAE : 1.4685030559590626
MSE : 3.37684018038163
RMSE : 1.8376180725008204
RMSLE : 0.35268030940555956
```

*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

기상청

V. 모형 검증

1. 변수 중요도 파악
2. 최종 모형 선택
3. 모형 성능 및 예측력 파악

1. 변수 중요도 파악 (1/2)

- **생성된 모델에 대해 가장 많은 영향을 주는 변수 중요도 파악**

```
#============================================================
# 변수 중요도 파악(표)
#============================================================
feature_list = pd.DataFrame({"importances":Traffic_GBM_Model.feature_importances_,
                             "feature_names":X.columns.tolist()})
feature_list = feature_list.sort_values("importances", ascending=False)
feature_list["scaled_importance"] =  feature_list["importances"] /
                                                    feature_list["importances"].values[0]
feature_list = feature_list[["feature_names", "scaled_importance",
                                                    "importances"]].reset_index(drop=True)
feature_list
```

  - **Traffic_GBM_Model.feature_importances_를 통해 모델의 변수에 따른 중요도를 DataFrame으로 확인**

> 실행 결과

	feature_names	scaled_importance	importances
0	RN_3_MAX	1.000000	0.339127
1	WS_AVG_2_STD	0.912022	0.309291
2	TA_MAX_2_STD	0.314214	0.106558
3	WS_MAX_6_STD	0.256017	0.086822
4	WS_MAX_2_MIN	0.175448	0.059499
5	TA_MIN_6_STD	0.106864	0.036240
6	hour	0.068391	0.023193
7	AVG_SPEED	0.034621	0.011741
8	TA_MIN_6_MAX	0.028488	0.009661
9	month	0.015330	0.005199
10	JD_3KM_STD	0.006766	0.002295
11	GISID	0.005586	0.001894
12	PM_3KM_STD	0.004729	0.001604
13	CONZONE	0.004270	0.001448
14	VOLUME_ALL	0.004184	0.001419
15	KS_3KM_AVG	0.003739	0.001268
16	BY_3KM_AVG	0.003586	0.001216
17	BY_3KM_STD	0.002909	0.000986
18	JD_1KM_AVG	0.001345	0.000456
19	BUSlane	0.000237	0.000080
20	SPEEDLIMIT	0.000000	0.000000

*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

1. 변수 중요도 파악 (2/2)

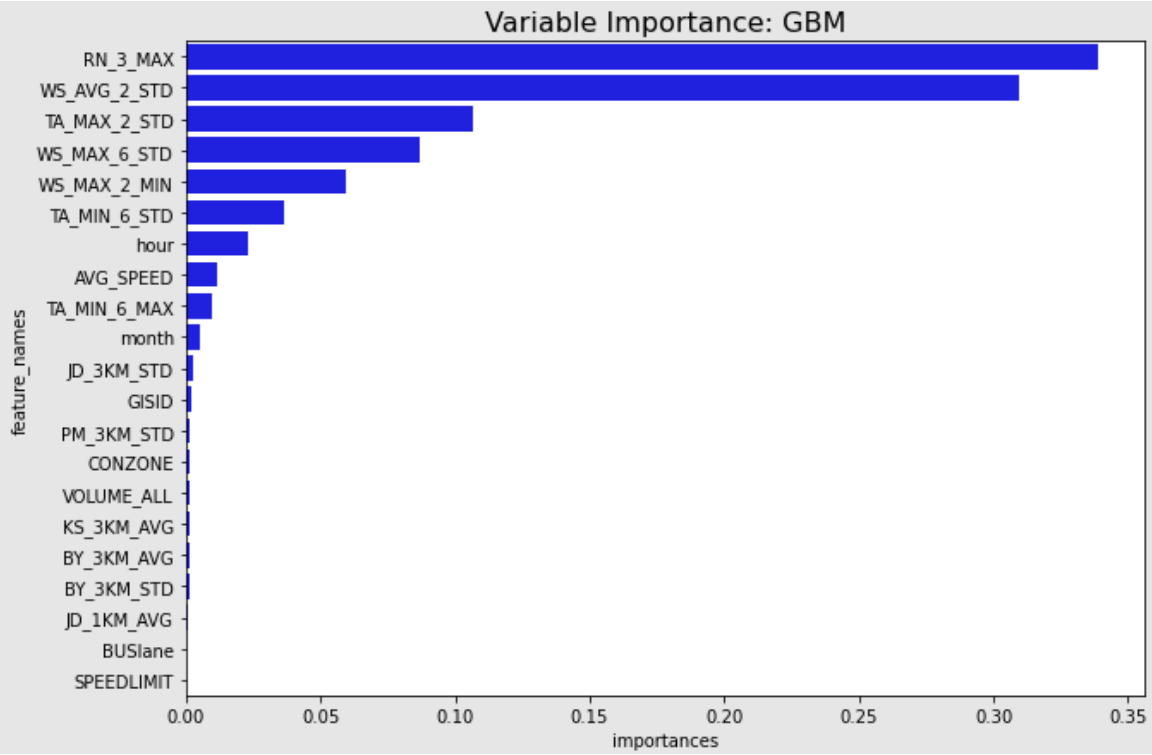
- **생성된 모델에 대해 가장 많은 영향을 주는 변수 중요도 시각화**

```
# 변수 중요도 파악(그래프)
plt.figure(figsize=(10, 7))
plt.title("Variable Importance: GBM",fontsize=16)
sns.barplot(data=feature_list, x="importances", y="feature_names", color="b")
```

  - **Traffic_GBM_Model.feature_importances_를 통해 모델의 변수에 따른 중요도를 seaborn의 barplot으로 시각화**

> 실행 결과

<AxesSubplot:title={"center":"Variable Importance: GBM"}, xlabel="importances", ylabel="feature_names">



*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

2. 최종 모형 선택 (1/3)

- **모형 입력변수 선택**

```
#============================================================
# 모형 입력변수 선택
#============================================================
i_index = []
rmse = []
Traffic_GBM_Model_feature = Traffic_GBM_Model
for i in [3, 5, 7, 9]: # 3, 5, 7, 9개의 변수 선택 위해 반복문 사용
    cols_to_set = ["CONZONE", "GISID", "month", "hour"]
    cols = cols_to_set + list(feature_list.feature_names[:i])

    train_feature = X_train[cols]
    valid_feature = X_valid[cols]

    Traffic_GBM_Model_feature.fit(train_feature, y_train)  # training_frame = train_feature
    pred = Traffic_GBM_Model_feature.predict(valid_feature)  # validation_frame = valid_feature

    i_index.append(i)
    rmse.append(np.sqrt(np.sum((y_valid - pred) ** 2) / len(pred)))
    print(i, "RMSE :", np.sqrt(np.sum((y_valid - pred) ** 2) / len(pred)))
```

  - For 반복문을 이용하여 3, 5, 7, 9개의 변수 선택
    - 반복문으로 생성된 각각의 train_feature, valid_feature 변수 생성
    - Traffic_GBM_Model을 이용하여, 학습 데이터로 데이터를 학습시키고 검증 데이터(valid)로 예측하여 i_index와 rmse 정확도 기록

> 실행 결과

```
3 RMSE : 1.8844998829944104
5 RMSE : 1.8500287673149127
7 RMSE : 1.8430637245154182
9 RMSE : 1.838649680490253
```

*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

2. 최종 모형 선택 (2/3)

- **RMSE 비교를 위해 최종적으로 feature_test라는 DataFrame 생성**

```
#============================================================
# Model rmse
#============================================================
feature_test = pd.DataFrame({"RMSE":rmse}, index=i_index).sort_values("RMSE",
                                                                      ascending=True)
feature_test
```

  - 변수별 RMSE 결과
    - 상위 변수 3개 : 1.884500
    - 상위 변수 5개 : 1.850029
    - 상위 변수 7개 : 1.843064
    - 상위 변수 9개 : 1.838650
  - 따라서 상위 변수 9개를 사용하여 최종 모델 선정
    - RN_3_MAX : 이전 3시간 동안 강수의 최대
    - WS_AVG_2_STD : 이전 2시간 동안 평균 풍속의 편차
    - TA_MAX_2_STD : 이전 2시간 동안 최대 기온의 편차
    - WS_MAX_6_STD : 이전 6시간 동안 최대 풍속의 편차
    - WS_MAX_2_MIN : 이전 2시간 동안 최대 풍속의 최소
    - TA_MIN_6_STD : 이전 6시간 동안 최소 기온의 편차
    - hour : 사고 발생 시간
    - AVG_SPEED : 1시간 평균 속도
    - TA_MIN_6_MAX : 이전 6시간 동안 최소 기온의 최대

> 실행 결과

	RMSE
9	1.838650
7	1.843064
5	1.850029
3	1.884500

*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

2. 최종 모형 선택 (3/3)

```
#============================================================
# 최종 모형 선택
#============================================================
best_i = feature_test.index[0]
cols = cols_to_set + list(feature_list.feature_names[:best_i])

train_feature = X_train[cols] # train_feature <- train
test_feature = X_test[cols]  # test_feature <- test

# create Model
Traffic_GBM_Model_final = Traffic_GBM_Model

# Model Summary
Traffic_GBM_Model_final
```

> 실행 결과

```
GradientBoostingRegressor(max_depth=15, max_features=0.51,
                          min_samples_leaf=0.075, n_estimators=148,
                          random_state=1234, subsample=0.85)
```

*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

3. 모형 성능 및 예측력 파악 (1/2)

- **모형 성능 및 예측력 파악**
  - **테스트 데이터를 예측하여 정확도 결과 확인**

```
#============================================================
# 모형 성능
#============================================================
# Predict on test set
Traffic_GBM_Model_final.fit(train_feature, y_train)
pred = Traffic_GBM_Model_final.predict(test_feature)

print("MODEL : Traffic_GBM_Model_final")
print("MAE :", np.mean(abs(y_test - pred)))
print("MSE :", np.sum((y_test - pred) ** 2) / len(pred))
print("RMSE :", np.sqrt(np.sum((y_test - pred) ** 2) / len(pred)))
print("RMSLE :", np.sqrt(np.mean(np.power(np.log1p(pred) - np.log1p(y_test), 2))))
```

> 실행 결과

```
MODEL : Traffic_GBM_Model_final
MAE : 1.4746798673602468
MSE : 3.4037872141292573
RMSE : 1.8449355582592193
RMSLE : 0.35537152046692116
```

*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

3. 모형 성능 및 예측력 파악 (2/2)

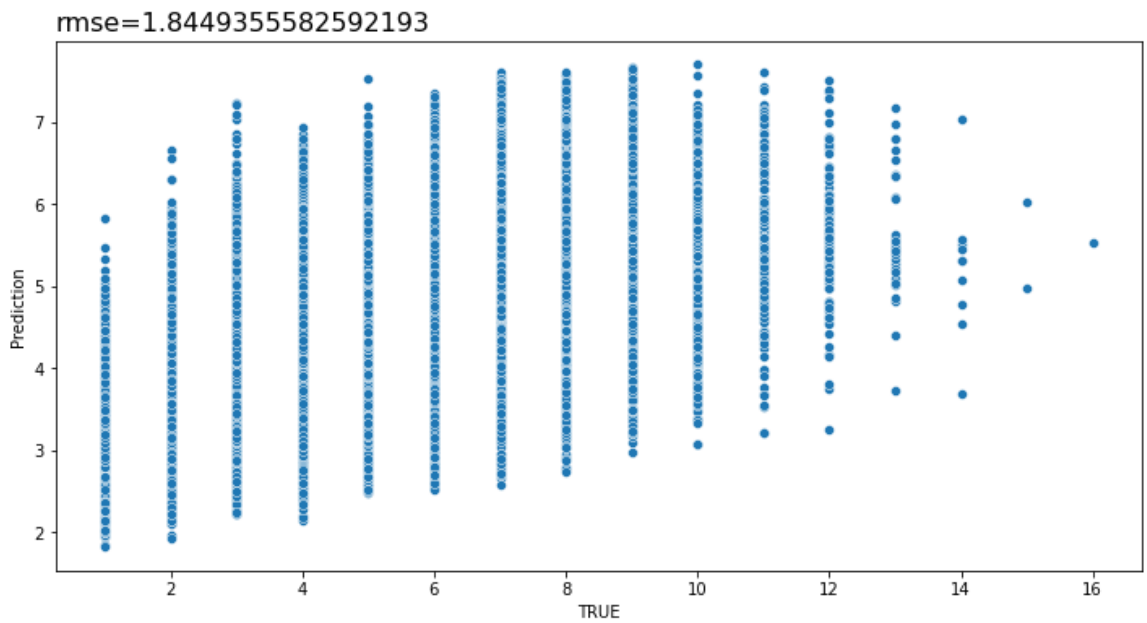
- **RMSE 그래프 파악**

```
#=============================================================
# RMSE
#=============================================================
Pred_conversion = pd.DataFrame({"pred_col":pred, "y_test_col":y_test}).set_index("y_test_col")
RMSE = np.sqrt(np.sum((y_test - pred) ** 2) / len(pred))
plt.figure(figsize=(12, 6))
sns.scatterplot(data=Pred_conversion, x=Pred_conversion.index, y="pred_col")
plt.title("rmse =" + str(RMSE), fontsize=16, loc="left" )
plt.xlabel("TRUE")
plt.ylabel("Prediction")
```

  - **seaborn에 scatterplot을 사용해 RMSE 그래프 파악**

> 실행 결과

```
Text(0, 0.5, 'Prediction')
```



*** Python과 R의 랜덤 효과가 상이하여 모형 결과가 다를 수 있습니다.**

본 문서의 내용은 기상청 날씨마루(http://big.kma.go.kr)의
분석 플랫폼 활용을 위한 Python 프로그래밍 교육 자료입니다.