

In [7]:

```

import nltk
import matplotlib
import numpy as np
import pandas as pd
from PIL import Image
from konlpy.tag import Okt, Mecab
from collections import Counter
from nltk.corpus import stopwords
import requests, os, re, time, json
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import set_matplotlib_formats
matplotlib.rcParams['font.family'] = 'Malgun Gothic'
set_matplotlib_formats('retina')
matplotlib.rcParams['axes.unicode_minus'] = False

```

In [15]:

```

site = 'https://openapi.naver.com/v1/search/news.json'

params = {
    'query' : '고속도로',
    'start' : 1,
    'display' : 100,
    'sort' : 'sim'}

headers = {
    'X-Naver-Client-Id' : '_HV4Mav6gAsfgons6mTA',
    'X-Naver-Client-Secret' : 'esqEqz0GLv'}

while True :
    time.sleep(1)
    data_dict = {
        'title' : [],
        'description' : []
    }
    print(f'{params["start"]} 수집중')
    res = requests.get(site, params=params, headers=headers)
    json_root = json.loads(res.text)
    items = json_root['items']

    for item in items :
        title = item['title']
        description = item['description']
        data_dict['title'].append(title)
        data_dict['description'].append(description)

    df1 = pd.DataFrame(data_dict)

    if os.path.exists('고속국도.csv') == False :
        df1.to_csv('고속국도.csv', encoding='utf-8-sig', index=False)
    else :
        df1.to_csv('고속국도.csv', encoding='utf-8-sig', index=False, header=False, mode='a')

    Start = json_root['start']
    Start += 100
    if Start < 1000 :
        params['start'] = Start
    else :
        break

print('수집완료')

```

1 수집중  
101 수집중  
201 수집중  
301 수집중  
401 수집중  
501 수집중  
601 수집중  
701 수집중  
801 수집중  
901 수집중  
수집완료

In [17]:

```

df = pd.read_csv('고속국도.csv')
df.head()

```

Out[17]:

	title	description
0	인천공항<b>고속도로</b> 차량 <b>사고</b>후 화재...음주여부 조사	인천공항<b>고속도로</b>에서 재규어 차량을 몰던 30대가 빗길에 미끄러져 중앙

1	美 당국, 테슬라 '오토파일럿' 추돌<b>사고</b> 11건 조사 착수	18년부터 11건<b>사고</b>로 1명 사망, 17명 부상 '완전한 자율운행으로 ...
2	인천공항<b>고속도로</b> 승용차 <b>사고</b>로 전소...음주운전 정황 확인 중	오늘(17일) 새벽 1시 50분쯤 인천국제공항<b>고속도로</b> 서울 방면 김포공...
3	미 교통당국, 테슬라 자율주행 충돌 <b>사고</b> 11건 조사 착수	미국 교통안전 규제당국이 전기차 테슬라의 오토파일럿(자율주행)과 연관된 11건의 충...
4	<b>고속도로</b> 승용차-화물차 충돌...교통<b>사고</b> 잊따라	밤사이 사건<b>사고</b>, 임상재 기자입니다. ◀ 리포트 ▶ 화물차 앞부분이 형...

```
In [18]: def text_cleaning(text) :
    hangul = re.compile('[^ㄱ-ㅣ가-힣]+')
    result = hangul.sub('', str(text))
    return result

df['title']      = df['title'].apply(lambda x : text_cleaning(x))
df['description'] = df['description'].apply(lambda x : text_cleaning(x))

title_corpus      = ''.join(df['title'].tolist())
description_corpus = ''.join(df['description'].tolist())
```

```
In [19]: # tagger = Okt()
tagger = Mecab('C:\Mecab\mecab-ko-dic')
title_nouns = tagger.nouns(title_corpus)
description_nouns = tagger.nouns(description_corpus)

title_count = Counter(title_nouns)
description_count = Counter(description_nouns)

with open('korean_stopwords.txt', encoding='utf-8') as fp :
    stopwords = fp.readlines()

stopwords = [x.strip() for x in stopwords]

title_dict = {}
description_dict = {}

for key in title_count :
    if len(key) > 1 :
        title_dict[key] = title_count[key]

for key in description_count :
    if len(key) > 1 :
        description_dict[key] = description_count[key]

remove_title_count = Counter(title_dict)
remove_description_count = Counter(description_dict)

title_dict = {}
for key in remove_title_count :
    if key not in stopwords :
        title_dict[key] = remove_title_count[key]

description_dict = {}
for key in remove_description_count :
    if key not in stopwords :
        description_dict[key] = remove_description_count[key]

remove_title_count = Counter(title_dict)
remove_description_count = Counter(description_dict)
```

```
In [20]: del remove_title_count['고속국도']
del remove_description_count['고속국도']
```

```
In [22]: wc = WordCloud(stopwords=spwords, font_path="c:/Windows/Fonts/malgun.ttf", background_color='white', width=500, height=500)
wc.generate_from_frequencies(dict(tag))
plt.figure(figsize=(12, 12))
plt.imshow(wc, interpolation="bilinear")
plt.axis('off')
plt.show()
```





Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

# ITS 국가 교통 정보 센터

- <https://www.its.go.kr/opendata/opendataList?service=event>
- 공사 사고 정보

In [14]:

```
import re
import os
import pandas as pd
from zipfile import ZipFile
from IPython.display import clear_output
```

In [15]:

```
# zip파일의 파일을 추출할 경로,
extract_path = './2018_csv_file/'

if not os.path.isdir(extract_path) :
    os.mkdir(extract_path)

# 0000년도 zip 파일에서 extract_path에 압축된 폴더 추출,
for root, dirs, files in os.walk(f'2020/'):
    for file_name in files:
        ZipFile(f'2020/{file_name}').extractall(extract_path)

# 압축 폴더로부터 csv 파일의 명칭을 리스트에 담기.
file_name = []
for r, d, fi in os.walk(f'{extract_path}') :
    for a in fi :
        file_name.append(a)

# 빌드 명칭만 가져오기.
variable_name = [f.split('.')[0].split('_')[-1] for f in file_name]

# for f_name, v_name in zip(file_name, variable_name) :
#     globals()[f'df_{v_name}'] = pd.read_csv(f'test/{f_name}', error_bad_lines=False, warn_bad_lines = False , se
```

In [25]:

```
df_all = pd.DataFrame()
for file in file_name :
    df = pd.read_csv(f'{extract_path}/{file}', error_bad_lines=False,
                     warn_bad_lines = False, sep = ',', encoding='cp949', header=None)
    df_all = pd.concat([df_all, df])

# 결합하였으니 인덱스 재설정.
df_all.index = list(range(df_all.shape[0]))
```

In [26]:

```
df2 = df_all.iloc[:, [0, 4, 5, 6, 7, 8, 9, 10]]
df2.columns = ['date', 'num', 'event', 'address', 'lon', 'lat', 'nan1', 'nan2']
df2
```

Out[26]:

		date	num	event	address	lon	lat	nan1	nan2
0		2018-01-02 00:17:00	1.0	사고	다산로 청구역 약수역 1차로 주의운전 차량고장 -	127.013397	37.559435	NaN	NaN
1		2018-01-02 02:27:00	1.0	사고	서울외곽순환고속도로 서운JC 계양IC 1차로 차량고장 주의운전 -	126.751386	37.5356	NaN	NaN
2		2018-01-02 04:26:30	1.0	사고	창경궁로 중구청앞사거리 을지로4가 1차로 주의운전 교통사고 -	126.998509	37.566201	NaN	NaN
3		2018-01-02 06:22:01	1.0	사고	중부고속도로 남이천IC 일죽IC 1차로 차량사고 주의운전 -	127.44106	37.136445	NaN	NaN
4		2018-01-02 06:57:00	1.0	사고	석동로59번길 냉천3사거리 천지가앞사거리 1차로 주의운전 교통사고 -	128.703638	35.156766	NaN	NaN
...		...	...	...	...	...	...	...	...
120598		2018-12-31 22:05:30	1	사고	삼일대로 필동2가삼익주택 퇴계로2가 1차로 주의운전 교통사고 -	126.989626	37.561237	NaN	NaN
120599		2018-12-31 22:20:30	1	사고	강변북로 토평강변로연결 강변북로분기1 1차로 주의운전 교통사고 -	127.127702	37.570763	NaN	NaN
120600		2018-12-31 23:14:44	1	사고	[서해안선]영광부근(65K) (2차로) 승용차 관련 사고 처리중 2차로 차단 사고	126.5834	35.3571	NaN	NaN
120601		2018-12-31 23:25:31	1	사고	경부고속도로 판교JC 판교IC 5차로 차량고장 주의운전 -	127.100187	37.399173	NaN	NaN
120602		2018-12-31 23:53:20	1	사고	[청주영덕선]화서(휴)부근(46.6K) (갓길)화물차고장 처리중 고장	127.9061	36.4462	NaN	NaN

120603 rows × 8 columns

In [56]:

```
# 각 컬럼명 데이터를 원하는 방향으로 타입 변환.
df2['num'] = pd.to_numeric(df2['num'])
df2['event'] = [str(i).strip() for i in df2['event']]
df2['lon'] = ['한글' if str(i) == 'nan' else i for i in df2['lon']]
df2['lat'] = [0 if str(i) == 'nan' else i for i in df2['lat']]
df2['nan1'] = [0 if str(i) == 'nan' else i for i in df2['nan1']]
df2['nan2'] = [0 if str(i) == 'nan' else i for i in df2['nan2']]
```

<ipython-input-56-8bb0f7c339c7>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2['num'] = pd.to_numeric(df2['num'])
```

<ipython-input-56-8bb0f7c339c7>:2: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2['event'] = [str(i).strip() for i in df2['event']]
```

<ipython-input-56-8bb0f7c339c7>:3: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2['lon'] = ['한글' if str(i) == 'nan' else i for i in df2['lon']]
```

<ipython-input-56-8bb0f7c339c7>:4: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2['lat'] = [0 if str(i) == 'nan' else i for i in df2['lat']]
```

<ipython-input-56-8bb0f7c339c7>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2['nan1'] = [0 if str(i) == 'nan' else i for i in df2['nan1']]
```

<ipython-input-56-8bb0f7c339c7>:6: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2['nan2'] = [0 if str(i) == 'nan' else i for i in df2['nan2']]
```

In [57]:

```
df2.dropna(inplace=True)
```

<ipython-input-57-f66736151044>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
df2.dropna(inplace=True)
```

In [58]:

```
df2.index = list(range(df2.shape[0]))
```

In [59]:

```
# 일련 데이터로 인해, 특정 컬럼내에 한글이 존재하면 이에 해당하는 인덱스 반환.
index_list = []
for j, i in enumerate(df2['lon']) :
    a = re.sub('[^0-9.]', '#', str(i))
    if '#' in a :
        index_list.append(j)
    clear_output(wait = True)
    print(f'{j+1}/{df2.shape[0]}번째')
```

IOPub message rate exceeded.  
The notebook server will temporarily stop sending output  
to the client in order to avoid crashing it.  
To change this limit, set the config variable

```
--NotebookApp.iopub_msg_rate_limit`.
```

Current values:

NotebookApp.iopub\_msg\_rate\_limit=1000.0 (msgs/sec)

NotebookApp.rate\_limit\_window=3.0 (secs)

In [60]:

```
df2.drop(index_list, inplace=True)
```

C:\Users\user\anaconda3\lib\site-packages\pandas\core\frame.py:4308: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
    return super().drop()
```

In [61]:

```
df2.drop(['nan1', 'nan2'], inplace=True, axis=1)  
df2.isna().sum()
```

Out[61]:

```
date      0  
num       0  
event     0  
address   0  
lon       0  
lat       0  
dtype: int64
```

In [62]:

```
df2
```

Out[62]:

	date	num	event	address	lon	lat
0	2018-01-02 00:17:00	1.0	사고	다산로 청구역 약수역 1차로 주의운전 차량고장 -	127.013397	37.559435
1	2018-01-02 02:27:00	1.0	사고	서울외곽순환고속도로 서운JC 계양IC 1차로 차량고장 주의운전 -	126.751386	37.5356
2	2018-01-02 04:26:30	1.0	사고	창경궁로 중구청앞사거리 을지로4가 1차로 주의운전 교통사고 -	126.998509	37.566201
3	2018-01-02 06:22:01	1.0	사고	중부고속도로 납이천IC 일죽IC 1차로 차량사고 주의운전 -	127.44106	37.136445
4	2018-01-02 06:57:00	1.0	사고	석동로59번길 냉천3사거리 천지가앞사거리 1차로 주의운전 교통사고 -	128.703638	35.156766
...	...	...	...	...	...	...
120221	2018-12-31 22:05:30	1.0	사고	삼일대로 필동2가삼익주택 퇴계로2가 1차로 주의운전 교통사고 -	126.989626	37.561237
120222	2018-12-31 22:20:30	1.0	사고	강변북로 토평강변로연결 강변북로분기1 1차로 주의운전 교통사고 -	127.127702	37.570763
120223	2018-12-31 23:14:44	1.0	사고	[서해안선]영광부근(65K) (2차로) 승용차 관련 사고 처리중 2차로 차단 사고	126.5834	35.3571
120224	2018-12-31 23:25:31	1.0	사고	경부고속도로 판교JC 판교IC 5차로 차량고장 주의운전 -	127.100187	37.399173
120225	2018-12-31 23:53:20	1.0	사고	[청주영덕선]화서(휴)부근(46.6K) (갓길)화물차고장 처리중 고장	127.9061	36.4462

119284 rows × 6 columns

## 저장 및 확인

In [63]:

```
# 저장.  
df2.to_csv('2018.csv', encoding='cp949', index=False, header=True)
```

In [64]:

```
# 확인.  
pd.read_csv('2018.csv', encoding='cp949')
```

Out[64]:

	date	num	event	address	lon	lat
0	2018-01-02 00:17:00	1.0	사고	다산로 청구역 약수역 1차로 주의운전 차량고장 -	127.013397	37.5594348
1	2018-01-02 02:27:00	1.0	사고	서울외곽순환고속도로 서운JC 계양IC 1차로 차량고장 주의운전 -	126.751386	37.5355995023003
2	2018-01-02 04:26:30	1.0	사고	창경궁로 중구청앞사거리 을지로4가 1차로 주의운전 교통사고 -	126.998509	37.5662011
3	2018-01-02 06:22:01	1.0	사고	중부고속도로 납이천IC 일죽IC 1차로 차량사고 주의운전 -	127.441060	37.1364446890986
4	2018-01-02 06:57:00	1.0	사고	석동로59번길 냉천3사거리 천지가앞사거리 1차로 주의운전 교통사고 -	128.703638	35.156765684386
...	...	...	...	...	...	...

119279	2018-12-31 22:05:30	1.0	사고	삼일대로 필동2가삼익주택 퇴계로2가 1차로 주의운전 교통사고 -	126.989626	37.5612375
119280	2018-12-31 22:20:30	1.0	사고	강변북로 토펑강변로연결 강변북로분기1 1차로 주의운전 교통사고 -	127.127702	37.5707632
119281	2018-12-31 23:14:44	1.0	사고	[서해안선]영광부근(65K) (2차로) 승용차 관련 사고 처리중 2차로 차단 사고	126.583400	35.3571
119282	2018-12-31 23:25:31	1.0	사고	경부고속도로 판교JC 판교IC 5차로 차량고장 주의운전 -	127.100187	37.3991730651403
119283	2018-12-31 23:53:20	1.0	사고	[청주영덕선]화서(휴)부근(46.6K) (갓길)화물차고장 처리중 고장	127.906100	36.4462

119284 rows × 6 columns

## 2018 ~ 2019년 사고 데이터

### 2018년 데이터

```
In [70]: df = pd.read_csv('2018.csv', encoding='cp949')
df2 = df[df['event'] == '사고']
df2.index = list(range(df2.shape[0]))
df2.to_csv('2018_사고.csv', encoding='cp949', index=False, header=True)
print(f'{df2.shape[0]}개')
```

50941개

### 2019년 데이터

```
In [71]: df = pd.read_csv('2019.csv', encoding='cp949')
df2 = df[df['event'] == '사고']
df2.index = list(range(df2.shape[0]))
df2.to_csv('2019_사고.csv', encoding='cp949', index=False, header=True)
print(f'{df2.shape[0]}개')
```

63488개

### 2020년 데이터

```
In [72]: df = pd.read_csv('2020.csv', encoding='cp949')
df2 = df[df['event'] == '사고']
df2.index = list(range(df2.shape[0]))
df2.to_csv('2020_사고.csv', encoding='cp949', index=False, header=True)
print(f'{df2.shape[0]}개')
```

88734개

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

## Google Map API 활용

- <https://cloud.google.com/maps-platform/pricing?hl=ko>
- 응급실 주소를 활용한 위도 경도 계산.

In [4]:

```
import os
import sys
import time
import datetime
import requests
import pandas as pd
from IPython.display import clear_output
```

In [5]:

```
df = pd.read_csv('data.csv')
map_list = df.iloc[:,1]

lat_List, lon_List = [], []

Start = time.time()

for idx, name in enumerate(map_list) :
    URL = f'https://maps.googleapis.com/maps/api/geocode/json?key={Input your Google Api Key}\n        sensor=false&language=ko&address={name}'
    response = requests.get(URL)
    data = response.json()
    lat = data['results'][0]['geometry']['location']['lat']
    lon = data['results'][0]['geometry']['location']['lng']
    lat_List.append(lat)
    lon_List.append(lon)
    # time.sleep(1)
    clear_output(wait = True)
    print(f'{idx + 1}/{len(map_list)}번째 수집중')

df1 = df.iloc[:, [0, 1]]
df1['lat'] = lat_List
df1['lon'] = lon_List

data_dir = './data/'
if not os.path.isdir(data_dir) :
    os.mkdir(data_dir)
if os.path.exists(data_dir + 'data_응급실.csv') == False :
    df1.to_csv(data_dir + 'data_응급실.csv', encoding='utf-8-sig', index=False)
else :
    df1.to_csv(data_dir + 'data_응급실.csv', encoding='utf-8-sig', index=False, header=False, mode='a')

End = time.time()

print(f'수집완료({datetime.timedelta(seconds = End - Start)})')
```

234/234번째 수집중

수집완료(0:02:30.172377)

In [6]:

```
pd.read_csv('./data/data_응급실.csv')
```

Out[6]:

	병원명	주소	lat	lon
0	인제대학교 해운대백병원	부산광역시 해운대구 해운대로 875 (좌동)	35.173493	129.181979
1	의료법인 인당의료재단 해운대부민병원	부산광역시 해운대구 해운대로 584 (우동)	35.161369	129.155844
2	재단법인천주교부산교구유지재단 메리놀병원	부산광역시 중구 종구로 121 (대청동4가)	35.106164	129.030002
3	영도병원	부산광역시 영도구 태종로 85 (대교동2가)	35.092442	129.038617
4	의료법인 행도의료재단 해동병원	부산광역시 영도구 태종로 133 (봉래동3가)	35.094470	129.044345
...	...	...	...	...
229	한마음병원	제주특별자치도 제주시 연신로 52 (이도이동)	33.496352	126.546238
230	제주대학교병원	제주특별자치도 제주시 아란13길 15 (아라일동)	33.467043	126.545756
231	의료법인 혜인의료재단 한국병원	제주특별자치도 제주시 서광로 193 (삼도일동)	33.500382	126.516784
232	제주한라병원	제주특별자치도 제주시 도령로 65, (연동)	33.489820	126.485193
233	제주특별자치도서귀포의료원	제주특별자치도 서귀포시 장수로 47 (동홍동)	33.256096	126.563064

234 rows × 4 columns

# 사고가 많이 발생하는 고속국도에서의 골든 타임을 확보하기 위한 병원 입지 선정

## 사용 모듈

```
In [1]:  
import mglearn  
import calplot  
import datetime  
import platform  
import numpy as np  
import pandas as pd  
import seaborn as sb  
from tqdm import tqdm  
from scipy import stats  
import sklearn.metrics as m  
import statsmodels.api as sm  
from itertools import chain, combinations  
# pd.set_option('display.max_columns', None)  
import matplotlib  
import matplotlib.pyplot as plt  
matplotlib.rcParams['font.family'] = 'Malgun Gothic'  
matplotlib.rcParams['axes.unicode_minus'] = False  
import plotly.graph_objs as go  
  
from collections import Counter  
import matplotlib.pyplot as plt  
from IPython.display import Image  
  
from sklearn.model_selection import KFold  
from sklearn.pipeline import make_pipeline  
from sklearn.neural_network import MLPRegressor  
from sklearn.model_selection import GridSearchCV  
from sklearn.neighbors import KNeighborsRegressor  
from sklearn.compose import make_column_transformer  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler, MinMaxScaler  
from statsmodels.graphics.tsaplots import acf, plot_acf, plot_pacf  
from sklearn.ensemble import VotingClassifier, RandomForestRegressor  
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder  
from statsmodels.stats.outliers_influence import variance_inflation_factor  
from sklearn.linear_model import LinearRegression, Ridge, SGDRegressor, Lasso, ElasticNet  
  
import warnings  
warnings.simplefilter('ignore')
```

## 분석 전 사고 방향

Step1. 고속국도 위에서 사고를 일으키는 요인에 대한 분석을 진행.

Step2. 사고가 발생한 발생지\_시도 또는 발생지\_시군구별에 대한 위험도 계산.

- 즉, 사고 위험도에 대한 공식을 세우고(찾고), 각 행에 대한 위험도를 계산한 위험도 컬럼을 생성.
- 위험도에 대한 예측을 진행? 위험도에 대한 분류를 하는 것이 맞는가? <-- 머신러닝과 딥러닝
- 여기서 필요한 작업 -> 각 시군구 별 위경도 데이터 확보가 무조건 필요.

Step3. 각 분류별 응급실 + 휴게소 분포 현황에 대한 분석이 필요.

- 휴게소와 사고와의 관계가 끌까?
- 응급실과 사고와의 관계가 끌까?

Step4. 기타 고려 사항이 필요.

Step5. 위 스텝을 모두 고려한 시각화 진행.

## 데이터 읽기

```
In [2]:  
# 2017년 ~ 2019년 고속국도 내 사고 현황 데이터 불러오기.  
df = pd.read_csv('C:/Users/user/Desktop/express accident1.csv')  
df.shape
```

Out[2]: (12448, 34)

## 데이터 명세서

## - 변수(feature) 개수 : 36개

### • 발생일

- 2017년 ~ 2019년 Calmap을 통한 일별 사고 건수 표현. 일별 Risk 표현.
  - 일별, 월별, 요일에 따른 위험도 및 사고 건수 확인.
- 

### • 발생시간

- 0 ~ 24시 사이에서 발생한 사고 건수 및 시간대별 리스크 표현.
  - 막대 그래프(1시간 단위) dodge, 파이차트(출퇴근, 오전, 오후, 퇴근, 밤, 새벽)
- 

### • 발생지\_시도

- 권역별 사고건수 및 리스크 위주의 파악.
  - 막대그래프 dodge.
- 

### • 발생지\_시군구

- 지역별 사고건수 및 리스크 위주의 파악.
  - 막대그래프 dodge.
- 

### • 요일

- 요일별 사고건수 및 리스크 위주의 파악.
  - 막대그래프 dodge.
- 

### • 사고내용

- 중상, 사망 빈도 표현.
- 

### • 사고유형\_대분류

- 차대차, 차대 사람, 차량 단독
- 

### • 사고유형\_중분류

- 아래 항목에 대한 막대그래프 표현.
    - 공작물충돌
    - 길 가장자리 구역 통행중
    - 도로 이탈
    - 보도 통행중
    - 전도 : 자동차의 측면이 도로에 접한 상태로 넘어진 사고.
    - 전복 : 운행중의 사고로 인해 자동차가 뒤집어진 사고.
    - 추돌 : 주행 중 앞차가 급정거를 하고, 뒤쫓아가던 차가 부딪히는 사고.
    - 전도전복
    - 정면충돌
    - 차도통행중
    - 횡단중
    - 측면충돌
    - 후진중 충돌
    - 기타
- 

### • 사고유형

- 공작물 충돌에 대한 빈도 표현.
  - 막대그래프 dodge.
- 

### • 가해자법규위반

- 안전 의무 빈도 표현.
- 

### • 가해자성별

### • 피해자성별

- 가해자 및 피해자 안전, 의무 빈도 표현.
- 

### • 가해자연령

### • 피해자연령

- 가해자 및 피해자 연령별에 따른 리스크 및 사고건수 표현

## 중요 요소 변수

- 가해당사자종별
- 피해당사자종별
  - 빈도 표현.(보류)

- 가해자신체상해정도
- 피해자신체상해정도
  - 가해자 및 피해자 신체 상해 정도 비교 표현.

- 가해차량용도\_대분류
- 가해차량용도\_중분류
- 가해차량용도

- 피해차량용도\_대분류
- 피해차량용도\_중분류
- 피해차량용도
  - 6개중 차량 용도만을 사용하고, 2개만 사용.
  - 차량에 따른 발생지\_시군구별 사고건수 및 risk

- 도로종류
  - 고속 국도 데이터이므로 삭제.

- 도로형태\_대분류
- 도로형태
  - 삭제 예정

- 노면상태\_대분류
  - 포장, 비포장 라벨에 대한 데이터 불균형 문제로 삭제.

- 노면상태
- 기상상태
  - 두 데이터 간의 상관관계 plot 파악.
  - 기증치 임의 및 기준을 이용하여 RISK 계산시 활용.(분석이 필요해 보임)

- 사망자수
- 중상자수
- 경상자수
  - 시군구별에 따른 사망자, 중상자 수, 경상자 수 파악.
  - 발생시\_시군구 데이터와의 분석 및 결합.

- 부상신고자수
  - [https://www.index.go.kr/potal/main/EachDtlPageDetail.do?idx\\_cd=1614](https://www.index.go.kr/potal/main/EachDtlPageDetail.do?idx_cd=1614)
  - 교통사고 : \* 집계 기준 : 차의 교통으로 인하여 발생한 인적피해를 수반하는 경찰접수 처리한 교통사고 기준.
    - 교통사고 사망자 기준 : 교통사고 발생일로부터 30일 이내에 사망한 경우
    - 중상자 : 3주 이상의 치료를 요하는 부상자
    - 경상자 : 5일 이상 3주 미만의 치료를 요하는 부상자
    - 부상신고 : 5일 미만의 치료를 요하는 부상을 입은 경우

## 데이터 전처리

- 데이터 삭제
  - 사고 유형의 경우 사고유형\_중분류 데이터와의 중복으로 삭제.
  - 도로 종류는 고속국도로 한정시켜 가져온 데이터이므로 삭제.

In [3]:

```
# 필요없는 데이터 삭제.
df.drop(['도로형태', '도로형태_대분류', '도로종류', '사고유형', '노면상태_대분류'], axis = 1, inplace = True)
df.head()
```

Out[3]:

발생일	발생시	발생지_시군구	발생일	사고내	사고유형_대	사고유형_중	가해자법규위반	가해자성별	...	가해차량	피해차량_용도_대	피해차량_용도_중	피해차량	기상상	노면상	사망자	중상자	경상자	부상신고
-----	-----	---------	-----	-----	--------	--------	---------	-------	-----	------	-----------	-----------	------	-----	-----	-----	-----	-----	------

	간	시	도	용	분	류	분	류	용	도	분	류	분	류	용	도	태	태	수	수	수	수	자수	
0	20170101	00	시	경북	구미	시	일	중상	차대차	추돌	안전운전 의무 불이행	남	...	승용차	비사업용	승용차	승용차	맑음	건조	0	8	0	1	
1	20170101	02	시	충남	홍성	군	일	사망	차대차	추돌	안전운전 의무 불이행	남	...	승용차	비사업용	승용차	승용차	맑음	건조	1	0	1	0	
2	20170101	03	시	경북	상주시	시	일	경상	차대차	기타	안전운전 의무 불이행	남	...	위험운송	사업용	렌터카	렌터카	맑음	건조	0	0	2	0	
3	20170101	04	시	서울	금천	구	일	부상신고	차대차	측면충돌	기타	기타	불명	...	불명	사업용	렌터카	렌터카	맑음	젖음/습기	0	0	0	1
4	20170101	07	시	강원	강릉	시	일	경상	차대차	추돌	안전운전 의무 불이행	여	...	승용차	비사업용	승용차	승용차	맑음	건조	0	0	4	0	

5 rows × 29 columns

### 데이터 결측치 확인

- 결측치 확인 결과 결측은 존재하지 않음.

In [4]:

```
df.isna().sum()
```

Out[4]:

발생일	0
발생시간	0
발생지_시도	0
발생지_시군구	0
요일	0
사고내용	0
사고유형_대분류	0
사고유형_중분류	0
가해자법규위반	0
가해자성별	0
가해자연령	0
가해당사자종별	0
가해자신체상해정도	0
피해자성별	0
피해자연령	0
피해당사자종별	0
피해자신체상해정도	0
가해차량용도_대분류	0
가해차량용도_중분류	0
가해차량용도	0
피해차량용도_대분류	0
피해차량용도_중분류	0
피해차량용도	0
기상상태	0
노면상태	0
사망자수	0
중상자수	0
경상자수	0
부상신고자수	0

dtype: int64

### 시군구 별에 따른 응급실 수

In [5]:

```
df_응급실 = pd.read_csv('C:/Users/user/Desktop/응급실 최종.csv')
df_응급실.columns = ['이름', '주소', '시도', '시군구', 'lat', 'lon']
응급 = df_응급실['시군구'].value_counts()
응급_dict = {n : i for i, n in zip(응급.values, 응급.index)}
응급_count = [0 if str(응급_dict.get(i)) == 'None' else 응급_dict.get(i) for i in df['발생지_시군구']]
df['응급실 개수'] = 응급_count
```

### 시군구 별에 따른 인구 데이터

In [6]:

```
df_인구 = pd.read_csv('C:/Users/user/Desktop/도시지역_인구현황_시군구_20210813095003.csv', encoding='euc-kr')
df_인구.index = df_인구['소재지(시군구)별(2)']
df_인구1 = df_인구[['2017', '2018', '2019']]
df_인구2 = df_인구1.drop(['소계'])[:2:]
```

```
# 미주출구에 대한 결측이 존재하여, 존재하는 연도의 평균으로 대체.
df_인구2['2017'] = [412702.0 if str(i) == '-' else i for i in df_인구2['2017']]
df_인구2['2018'] = [np.nan if str(i) == '-' else i for i in df_인구2['2018']]
df_인구2['2019'] = [np.nan if str(i) == '-' else i for i in df_인구2['2019']]
```

```
df_인구3 = df_인구2.dropna()
df_인구3 = df_인구3.astype('int')
b = round(df_인구3.mean(axis=1))
```

```
인구_dic = {i : j for i, j in zip(b.index, b.values)}
인구_count = [0 if str(인구_dic.get(i)) == 'None' else 인구_dic.get(i) for i in df['발생지_시군구']]
df['시군구별_인구'] = 인구_count
```

## Risk 가중치 계산

- <http://taas.koroad.or.kr/TCFS/>

In [7]:

```
df['Risk'] = (df['사망자수'] * 12) + (df['중상자수'] * 3) + (df['경상자수'] * 1) + (df['부상신고자수'] * 1)
df.head()
```

Out[7]:

	발생일	발생시간	발생지_시도	발생지_시군구	요일	사고내용	사고유형_대분류	사고유형_중분류	가해자별규위반	가해자성별	...	피해차량용도	기상상태	노면상태	사망자수	중상자수	경상자수	부상신고자수	응급실개수	시군구별_인구	Risk
0	20170101	00시	경북	구미시	일	중상	차대차	추돌	안전운전의무불이행	남	...	승용차	맑음	건조	0	8	0	1	3	421012.0	25
1	20170101	02시	충남	홍성군	일	사망	차대차	추돌	안전운전의무불이행	남	...	승용차	맑음	건조	1	0	1	0	1	101025.0	13
2	20170101	03시	경북	상주시	일	경상	차대차	기타	안전운전의무불이행	남	...	렌터카	맑음	건조	0	0	2	0	2	100644.0	2
3	20170101	04시	서울	금천구	일	부상신고	차대차	측면충돌	기타	기타불명	...	렌터카	맑음	점/습기	0	0	0	1	1	233960.0	1
4	20170101	07시	강원	강릉시	일	경상	차대차	추돌	안전운전의무불이행	여	...	승용차	맑음	건조	0	0	4	0	4	213450.0	4

5 rows × 32 columns

In [8]:

```
df.to_csv('total_data.csv')
```

In [9]:

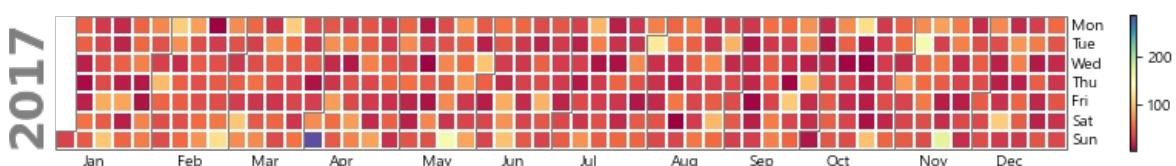
```
# a1 = df[['Risk', '발생지_시군구']]
# a2 = a1.groupby('발생지_시군구').sum()
# {n : i[0] for n, i in zip(a2.index, a2.values)}
```

발생일

In [10]:

```
data_2017 = pd.DataFrame()
df1 = df.copy()
df2 = df1[['발생일', 'Risk']]
df_2017_time = list(set([i for j, i in enumerate(df2['발생일']) if '2017' in str(i)]))
df2['발생일'] = df2['발생일'].astype('str')
df2['발생일'] = pd.to_datetime(df2['발생일'])
df2.index = df2['발생일']
df3 = df2[:4146].drop(['발생일'], axis = 1)
df4 = df3.groupby([df3.index.month, df3.index.day]).sum()
data_2017['Date'] = df_2017_time
data_2017['Risk'] = df4['Risk'].values
data_2017['Date'] = data_2017['Date'].astype('str')
data_2017.index = pd.to_datetime(data_2017['Date'])
calplot.calplot(data_2017['Risk'], cmap='Spectral', colorbar=True)
plt.show()
```

findfont: Font family ['Helvetica'] not found. Falling back to DejaVu Sans.



In [11]:

```

data_2018 = pd.DataFrame()
df1 = df.copy()
df2 = df1[['발생일', 'Risk']]
df_2018_time = list(set([i for j, i in enumerate(df2['발생일']) if '2018' in str(i)]))
df2['발생일'] = df2['발생일'].astype('str')
df2['발생일'] = pd.to_datetime(df2['발생일'])
df2.index = df2['발생일']
df3 = df2[416:8225].drop(['발생일'], axis = 1)
df4 = df3.groupby([df3.index.month, df3.index.day]).sum()
data_2018['Date'] = df_2018_time
data_2018['Risk'] = df4['Risk'].values
data_2018['Date'] = data_2018['Date'].astype('str')
data_2018.index = pd.to_datetime(data_2018['Date'])
calplot.calplot(data_2018['Risk'], cmap='YlGnBu', colorbar=True)
plt.show()

```

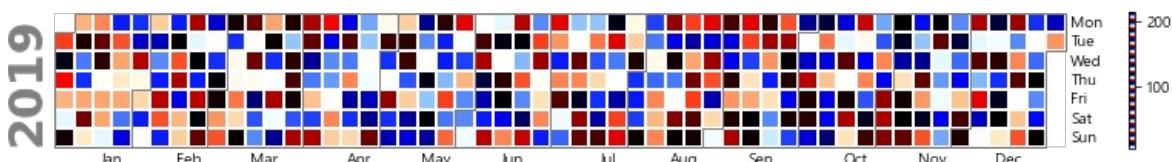


In [12]:

```

data_2019 = pd.DataFrame()
df1 = df.copy()
df2 = df1[['발생일', 'Risk']]
df_2019_time = list(set([i for j, i in enumerate(df2['발생일']) if '2019' in str(i)]))
df2['발생일'] = df2['발생일'].astype('str')
df2['발생일'] = pd.to_datetime(df2['발생일'])
df2.index = df2['발생일']
df3 = df2[8225:].drop(['발생일'], axis = 1)
df4 = df3.groupby([df3.index.month, df3.index.day]).sum()
data_2019['Date'] = df_2019_time
data_2019['Risk'] = df4['Risk'].values
data_2019['Date'] = data_2019['Date'].astype('str')
data_2019.index = pd.to_datetime(data_2019['Date'])
calplot.calplot(data_2019['Risk'], cmap='flag', colorbar=True)
plt.show()

```



## 발생시간

- 시간대별 사고 건수

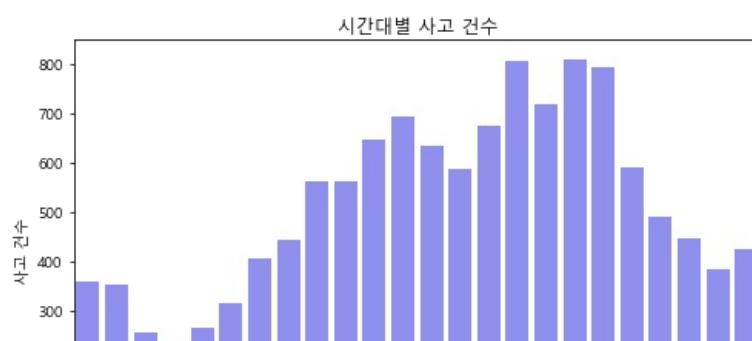
In [13]:

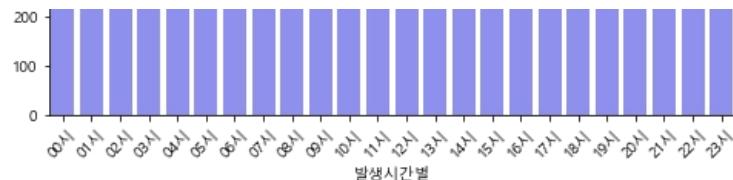
```

df_time = df[['발생시간','Risk']]
df_time_s = df_time.sort_values(by='발생시간')

plt.figure(figsize = (8, 5))
sb.countplot(data = df_time_s, x = '발생시간', color = 'blue', alpha=.5)
plt.xticks(rotation = 45)
plt.title('시간대별 사고 건수')
plt.xlabel('발생시간별')
plt.ylabel('사고 건수')
plt.show()

```

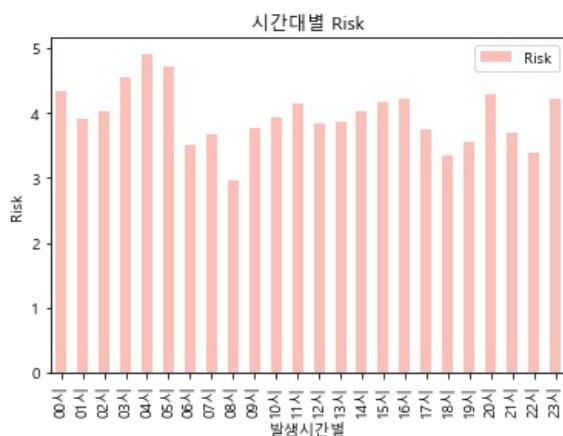




- 시간대별 Risk

In [14]:

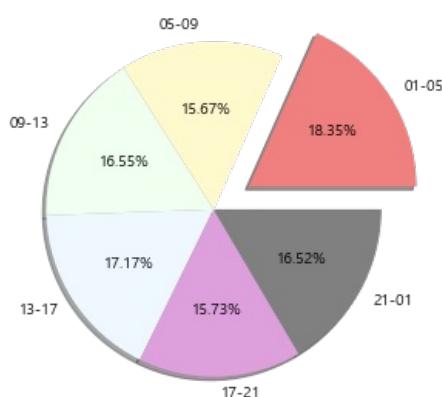
```
df_time_group = df_time_s.groupby('발생시간')
df_time_group = pd.DataFrame(df_time_group.mean()['Risk'])
df_time_group.plot(kind='bar', color='salmon', alpha=0.5)
plt.title('시간대별 Risk')
plt.xlabel('발생시간별')
plt.ylabel('Risk')
plt.show()
```



In [15]:

```
a = df_time_group.loc['01시':'04시'].mean()
b = df_time_group.loc['05시':'08시'].mean()
c = df_time_group.loc['09시':'12시'].mean()
d = df_time_group.loc['13시':'16시'].mean()
e = df_time_group.loc['17시':'20시'].mean()
f = df_time_group.loc[['21시', '22시', '23시', '00시']].mean()
date = pd.DataFrame()
date['01-05'] = a
date['05-09'] = b
date['09-13'] = c
date['13-17'] = d
date['17-21'] = e
date['21-01'] = f

col = ['lightcoral', 'lemonchiffon', 'honeydew', 'aliceblue', 'plum', 'grey']
labels = ['01-05', '05-09', '09-13', '13-17', '17-21', '21-01']
explode = [0.25, 0, 0, 0, 0, 0]
plt.figure(figsize=(8, 5))
plt.pie(date.T['Risk'].values, labels=labels, autopct='%.2f%%', shadow=True, explode=explode, colors=col)
plt.show()
```



In [16]:

```
df['발생시간'] = [i.split('시')[0] for i in df['발생시간']]
df['발생시간'] = df['발생시간'].astype('int64')
```

## 문자열 Encoding

- 문자열로 이루어진 데이터에 대한 인코딩 작업.
  - **Label encoding**
  - OneHot encoding
  - pd.get\_dummies
  - replace
  - make\_column\_transformer

### 발생지\_시도 encoding

- 강원, 경기, 경남, 경북, 광주, 대구, 대전, 부산, 서울, 세종, 울산, 인천, 전남, 전북, 충남, 충북
- 2017년 ~ 2019년 경기 인근 고속국도에서의 사고 건수가 압도적으로 많음.

```
In [17]: lbl1 = LabelEncoder()
Data_Label1 = lbl1.fit_transform( df[ '발생지_시도' ] )
Data_Label1
```

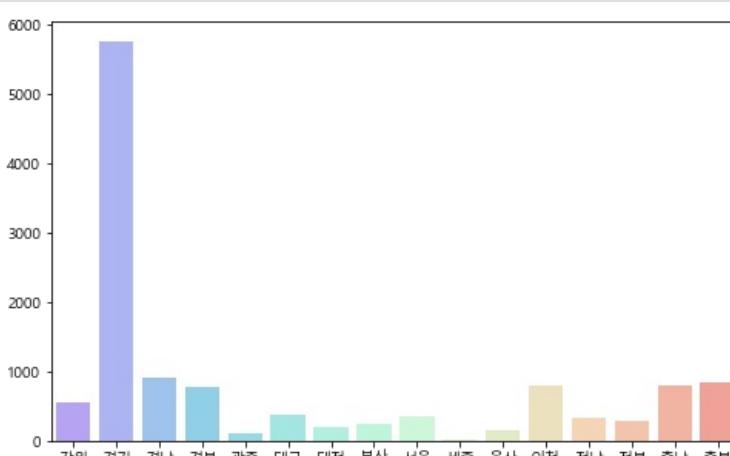
```
Out[17]: array([ 3, 14, 3, ..., 1, 1, 1])
```

```
In [18]: df_발생지_시도_dic = { idx : local for idx, local in enumerate(list(lbl1.classes_)) }
```

```
Out[18]: {0: '강원',
1: '경기',
2: '경남',
3: '경북',
4: '광주',
5: '대구',
6: '대전',
7: '부산',
8: '서울',
9: '세종',
10: '울산',
11: '인천',
12: '전남',
13: '전북',
14: '충남',
15: '충북'}
```

- 권역별 사고 건수 빈도

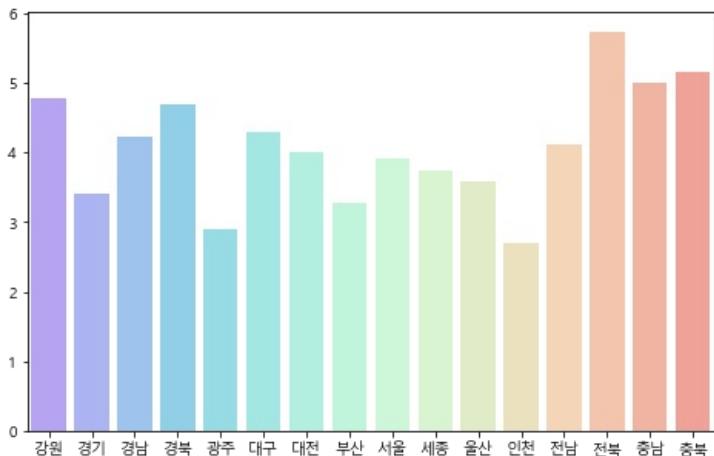
```
In [19]: local_list = [ df_발생지_시도_dic.get(i) for i in Data_Label1 ]
local_df = pd.DataFrame(local_list).value_counts()
b1 = [i[0] for i in local_df.index]
b2 = local_df.values
acc = pd.DataFrame()
acc['권역'] = b1
acc['사고건수'] = b2
acc1 = acc.sort_values(by='권역')
plt.figure(figsize=(8, 5))
sb.barplot(acc1['권역'].values, acc1['사고건수'].values, palette='rainbow', alpha=.5)
plt.show()
```



- 권역별 사고 리스크 Level

In [20]:

```
df_발생지_시도_risk = df[['발생지_시도', 'Risk']].groupby('발생지_시도').mean()
a1 = list(df_발생지_시도_risk.index)
a2 = [i[0] for i in df_발생지_시도_risk.values]
risk = pd.DataFrame()
risk['권역'] = a1
risk['Risk_mean'] = a2
plt.figure(figsize=(8, 5))
sb.barplot(risk['권역'].values, risk['Risk_mean'].values, palette='rainbow', alpha=.5)
plt.show()
```



In [21]:

```
df['발생지_시도'] = Data_Label1
```

#### 발생지\_시군구 encoding

- 총 148개의 지역에서 사고가 발생.
- 서울 용인시, 경기 성남시, 경기 화성시, 경기 평택시, 경남 김해시 순으로 사고 발생이 많았음.

In [22]:

```
lbl2 = LabelEncoder()
Data_Label2 = lbl2.fit_transform(df['발생지_시군구'])
Data_Label2
```

Out[22]: array([ 25, 155, 70, ..., 148, 157, 157])

In [23]:

```
df_발생지_시군구_dic = { idx : local for idx, local in enumerate(list(lbl2.classes_)) }
```

Out[23]: {0: '가평군',  
1: '강남구',  
2: '강동구',  
3: '강릉시',  
4: '강서구',  
5: '강진군',  
6: '거창군',  
7: '경산시',  
8: '경주시',  
9: '계룡시',  
10: '계양구',  
11: '고령군',  
12: '고성군',  
13: '고양시',  
14: '고창군',  
15: '곡성군',  
16: '공주시',  
17: '과천시',  
18: '광명시',  
19: '광산구',  
20: '광양시',  
21: '광주시',  
22: '괴산군',

23: '구례군',  
24: '구리시',  
25: '구미시',  
26: '군산시',  
27: '군위군',  
28: '군포시',  
29: '금산군',  
30: '금정구',  
31: '금천구',  
32: '기장군',  
33: '김제시',  
34: '김천시',  
35: '김포시',  
36: '김해시',  
37: '나주시',  
38: '남동구',  
39: '남양주시',  
40: '남원시',  
41: '노원구',  
42: '논산시',  
43: '단양군',  
44: '달서구',  
45: '달성군',  
46: '담양군',  
47: '당진시',  
48: '대덕구',  
49: '동구',  
50: '동해시',  
51: '목포시',  
52: '무안군',  
53: '무주군',  
54: '문경시',  
55: '미추홀구',  
56: '밀양시',  
57: '보령시',  
58: '보성군',  
59: '보은군',  
60: '부안군',  
61: '부여군',  
62: '부천시',  
63: '부평구',  
64: '북구',  
65: '사상구',  
66: '사천시',  
67: '사하구',  
68: '산청군',  
69: '삼척시',  
70: '상주시',  
71: '서구',  
72: '서산시',  
73: '서천군',  
74: '서초구',  
75: '성남시',  
76: '성북구',  
77: '성주군',  
78: '세종시',  
79: '속초시',  
80: '송파구',  
81: '수성구',  
82: '수원시',  
83: '순창군',  
84: '순천시',  
85: '시흥시',  
86: '안동시',  
87: '안산시',  
88: '안성시',  
89: '안양시',  
90: '양산시',  
91: '양양군',  
92: '양주시',  
93: '양천구',  
94: '양평군',  
95: '여주시',  
96: '연수구',  
97: '영광군',  
98: '영덕군',  
99: '영동군',  
100: '영암군',  
101: '영주시',  
102: '영천시',  
103: '예산군',  
104: '예천군',  
105: '오산시',

```
106: '옥천군',
107: '완주군',
108: '용인시',
109: '을주군',
110: '원주시',
111: '유성구',
112: '음성군',
113: '의성군',
114: '의왕시',
115: '의정부시',
116: '이천시',
117: '익산시',
118: '인제군',
119: '임실군',
120: '장성군',
121: '장수군',
122: '장흥군',
123: '전주시',
124: '정읍시',
125: '제천시',
126: '중구',
127: '중랑구',
128: '증평군',
129: '진안군',
130: '진주시',
131: '진천군',
132: '창녕군',
133: '창원시',
134: '천안시',
135: '청도군',
136: '청송군',
137: '청양군',
138: '청주시',
139: '춘천시',
140: '충주시',
141: '칠곡군',
142: '통영시',
143: '파주시',
144: '평창군',
145: '평택시',
146: '포천시',
147: '포항시',
148: '하남시',
149: '하동군',
150: '함안군',
151: '함양군',
152: '합평군',
153: '합천군',
154: '해운대구',
155: '홍성군',
156: '홍천군',
157: '화성시',
158: '화천군',
159: '횡성군'}
```

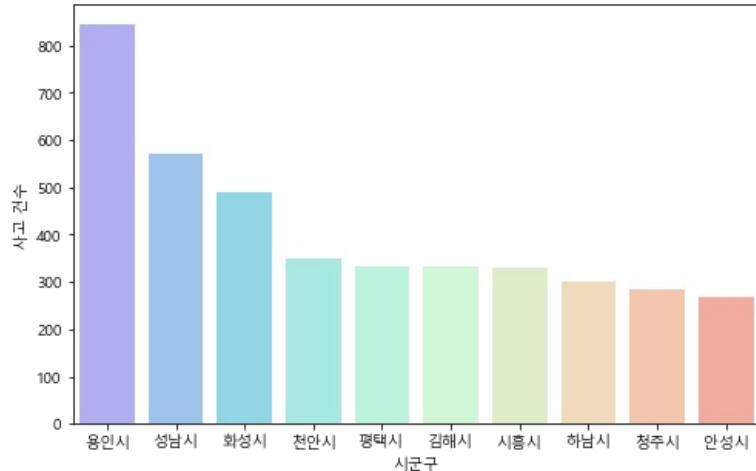
```
In [24]: local_list2 = [ df_발생지_시군구_dic.get(i) for i in Data_Label2 ]
local_df2 = pd.DataFrame(local_list2).value_counts()
local_df2[:10]
```

```
Out[24]: 용인시    844
성남시    570
화성시    490
천안시    349
평택시    333
김해시    333
시흥시    331
하남시    299
청주시    284
안성시    268
dtype: int64
```

- 사고 건수가 가장 많이 발생하는 지역 TOP 10

```
In [25]: plt.figure(figsize = (8, 5))
sb.barplot([i[0] for i in list(local_df2[:10].index)], local_df2[:10].values, palette='rainbow', alpha=.5)
plt.xlabel('시군구')
```

```
plt.ylabel('사고 건수')
plt.show()
```

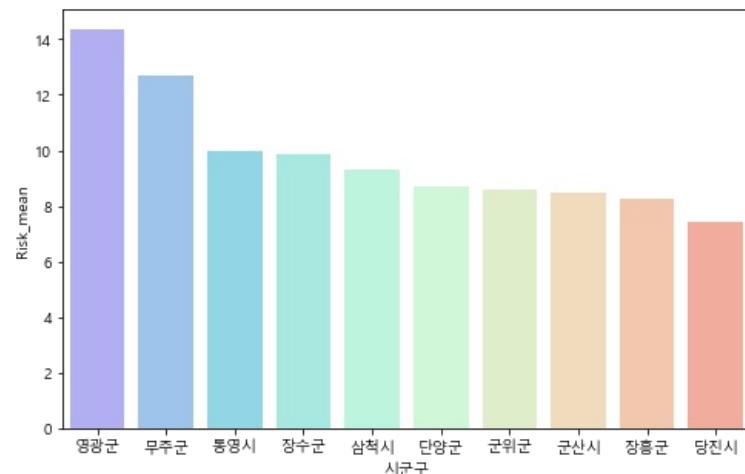


- 사고 리스크가 가장 큰 지역 TOP10

```
In [26]: df1 = df[['발생지_시군구', 'Risk']]
df1['발생지_시군구'] = ['창원시' if '통합' in i else i for i in df1['발생지_시군구']]
df2 = df1.groupby('발생지_시군구').mean()
df3 = df2.sort_values(by='Risk', ascending=False)[:10]

top10 = []
for i in list(df3.index):
    top10.append(i)

plt.figure(figsize = (8, 5))
sb.barplot(top10, [i[0] for i in df3.values], palette='rainbow', alpha=.5)
plt.xlabel('시군구')
plt.ylabel('Risk_mean')
plt.show()
```



```
In [27]: df['발생지_시군구'] = Data_Label2
```

### 요일 encoding

- 월, 화, 수, 목, 금, 토, 일
- 평일(월 ~ 금), 주말(토 ~ 일)
- 토요일, 금요일의 사고 빈도수가 가장 높음.
- 의외로 일요일 사고건수가 가장 낮게 나옴.

```
In [28]: lbl3 = LabelEncoder()
Data_Label3 = lbl3.fit_transform( df['요일'] )
Data_Label3
```

```
Out[28]: array([4, 4, 4, ..., 6, 6, 6])
```

```
In [29]: df_요일_dic = { idx : day for idx, day in enumerate(list(lbl3.classes_)) }
```

```
Out[29]: {0: '금', 1: '목', 2: '수', 3: '월', 4: '일', 5: '토', 6: '화'}
```

```
In [30]: b1 = [i[0] for i in local_df.index]
b2 = local_df.values
acc = pd.DataFrame()
acc['권역'] = b1
acc['사고건수'] = b2
acc1 = acc.sort_values(by='권역')
```

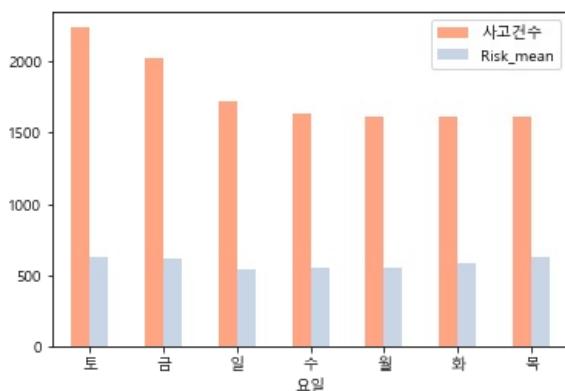
```
In [31]: local_list3 = [ df_요일_dic.get(i) for i in Data_Label3 ]
local_df3 = pd.DataFrame(local_list3).value_counts()

day = pd.DataFrame()
day['요일'] = [i[0] for i in local_df3.index]
day['사고건수'] = local_df3.values

# df[['요일', 'Risk']].groupby('요일').mean()
day['Risk_mean'] = [7.862475 * 80, 7.677824 * 80, 6.795964 * 80, 6.949886 * 80, 6.845606 * 80, 7.326139 * 80, 7.862475 * 80]
day.index = day['요일']

# plt.figure(figsize=(8, 5))
# sb.barplot([i[0] for i in list(local_df3.index)], local_df3.values, palette='rainbow', alpha=.5)
# plt.xlabel('요일')
# plt.ylabel('사고 건수')
# plt.show()

day.plot(kind='bar', y=['사고건수', 'Risk_mean'], color=['coral', 'lightsteelblue'], alpha=0.7, rot=0)
plt.show()
```



```
In [32]: df['요일'] = Data_Label3
```

사고내용 encoding

- 사망
- 중상

```
In [33]: lbl4 = LabelEncoder()
Data_Label4 = lbl4.fit_transform( df['사고내용'] )
Data_Label4
```

```
Out[33]: array([3, 2, 0, ..., 0, 0, 0])
```

```
In [34]: df_사고내용_dic = { idx : acc for idx, acc in enumerate(list(lbl4.classes_)) }
df_사고내용_dic
```

```
Out[34]: {0: '경상', 1: '부상신고', 2: '사망', 3: '중상'}
```

```
In [35]: local_list4 = [ df_사고내용_dic.get(i) for i in Data_Label4 ]
```

```

local_df4 = pd.DataFrame(local_list4).value_counts()
serious, death = local_df4.values[0], local_df4.values[-1]
total = serious + death
print(f'Results : \n Total Data : {total}\n Rate of death : {death} ({(death/total*100):.2f}%)\n')

```

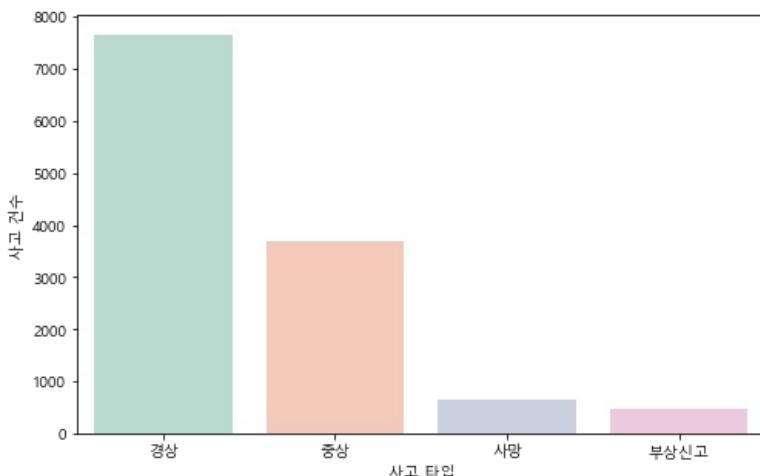
Results :  
 Total Data : 8122  
 Rate of death : 484 (5.96%)

In [36]:

```

plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df4.index)], local_df4.values, palette='Set2', alpha=.5)
plt.xlabel('사고 타입')
plt.ylabel('사고 건수')
plt.show()

```



In [37]:

```
df['사고내용'] = Data_Label4
```

### 사고유형\_대분류 encoding

- 차대 사람
- 차대차
- 차량 단도
- 차대차 사람의 비율이 압도적으로 높음. 즉, 고속국도 위에서 차와 차 사이에서 발생하는 사고가 가장 많음.

In [38]:

```

lbl5 = LabelEncoder()
Data_Label5 = lbl5.fit_transform(df['사고유형_대분류'])
Data_Label5

```

Out[38]:

```
array([1, 1, 1, ..., 1, 1, 1])
```

In [39]:

```
df['사고유형_대분류'] = Data_Label5
```

In [40]:

```

df_사고유형_대분류_dic = { idx : car for idx, car in enumerate(list(lbl5.classes_)) }
df_사고유형_대분류_dic

```

Out[40]:

```
{0: '차대사람', 1: '차대차', 2: '차량단독'}
```

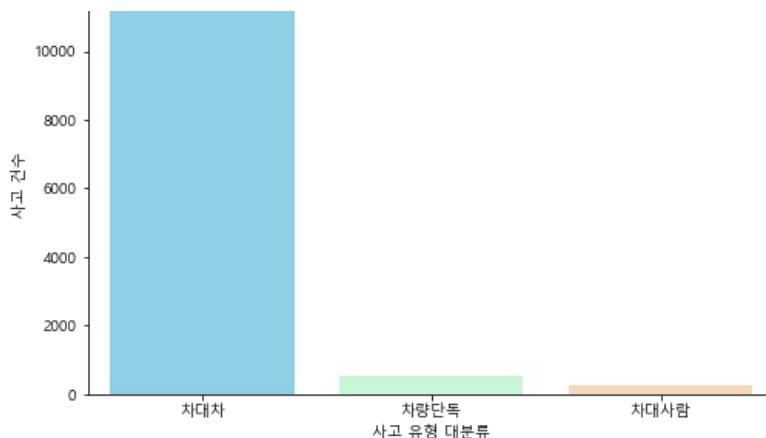
In [41]:

```

local_list5 = [ df_사고유형_대분류_dic.get(i) for i in Data_Label5 ]
local_df5 = pd.DataFrame(local_list5).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df5.index)], local_df5.values, palette='rainbow', alpha=.5)
plt.xlabel('사고 유형 대분류')
plt.ylabel('사고 건수')
plt.show()

```





### 사고유형\_중분류

- 공작물충돌
- 길 가장자리 구역 통행중
- 도로 이탈
- 보도 통행중
- 전도 : 자동차의 측면이 도로에 접한 상태로 넘어진 사고.
- 전복 : 운행중의 사고로 인해 자동차가 뒤집어진 사고.
- 추돌 : 주행 중 앞차가 급정거를 하고, 뒤쫓아가던 차가 부딪히는 사고.
- 전도전복
- 정면충돌
- 차도통행중
- 횡단중
- 측면충돌
- 후진중 충돌
- 기타
- 추돌 사고가 굉장히 많이 발생함. 기타 사고 비율도 두 번째로 높으나 기타가 정확히 어떤 사항인지는 알 수가 없음.
- 측면 충돌 사고의 비중 세번째로 높고, 나머지에 대한 사고는 거의 존재하지 않음.

```
In [42]: lbl6 = LabelEncoder()
Data_Label6 = lbl6.fit_transform( df['사고유형_중분류'] )
Data_Label6
```

```
Out[42]: array([11, 11, 1, ..., 1, 1, 1])
```

```
In [43]: df['사고유형_중분류'] = Data_Label6
```

```
In [44]: df_사고유형_중분류_dic = { idx : car for idx, car in enumerate(list(lbl6.classes_)) }
df_사고유형_중분류_dic
```

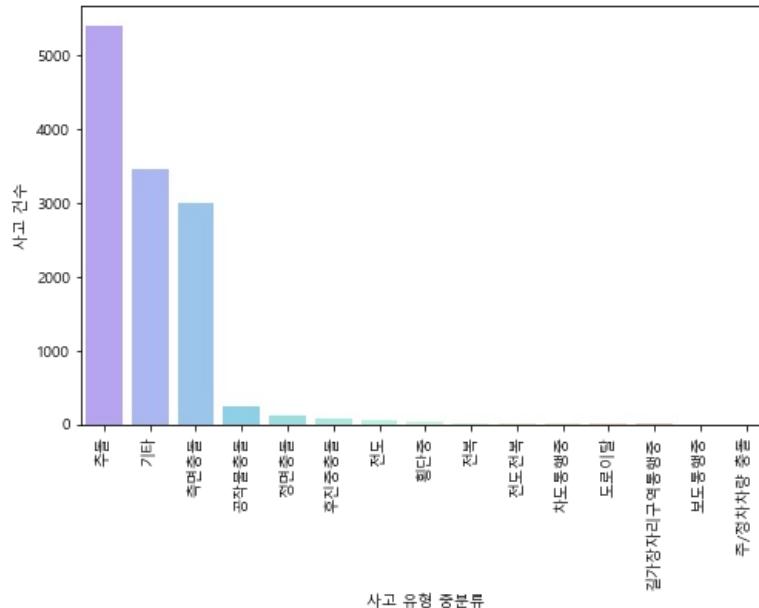
```
Out[44]: {0: '공작물충돌',
1: '기타',
2: '길가장자리구역통행중',
3: '도로이탈',
4: '보도통행중',
5: '전도',
6: '전도전복',
7: '전복',
8: '정면충돌',
9: '주/정차차량 충돌',
10: '차도통행중',
11: '추돌',
12: '측면충돌',
13: '횡단중',
14: '후진중충돌'}
```

```
In [45]: local_list6 = [ df_사고유형_중분류_dic.get(i) for i in Data_Label6 ]
local_df6 = pd.DataFrame(local_list6).value_counts()
plt.figure(figsize=(8, 5))
```

```

sb.barplot([i[0] for i in list(local_df6.index)], local_df6.values, palette='rainbow', alpha=.5)
plt.xticks(rotation = 90)
plt.xlabel('사고 유형 종분류')
plt.ylabel('사고 건수')
plt.show()

```



## 가해자법규위반

- 과속
- 신호 위반
- 안전거리 미확보
- 안전 운전 의무 불이행
- 중앙선 침범
- 안전 운전 의무 불이행의 비율이 가장 높고, 안전 거리 미확보의 비율도 상대적인 관점에서 적지 않음.

```

In [46]: lbl7 = LabelEncoder()
Data_Label7 = lbl7.fit_transform( df['가해자법규위반'] )
Data_Label7

```

```
Out[46]: array([5, 5, 5, ..., 5, 4, 4])
```

```
In [47]: df['가해자법규위반'] = Data_Label7
```

```
In [48]: df_가해자법규위반_dic = { idx : man for idx, man in enumerate(list(lbl7.classes_)) }
```

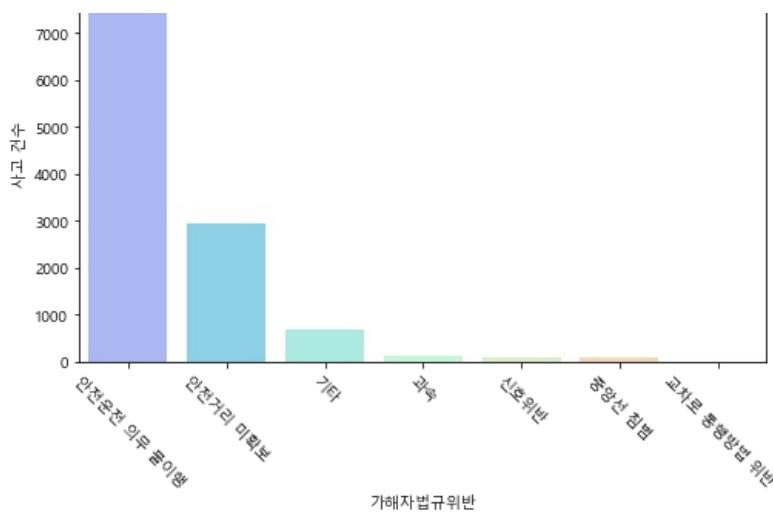
```
Out[48]: {0: '과속',
 1: '교차로 통행방법 위반',
 2: '기타',
 3: '신호위반',
 4: '안전거리 미확보',
 5: '안전운전 의무 불이행',
 6: '중앙선 침범'}
```

```

In [49]: local_list7 = [ df_가해자법규위반_dic.get(i) for i in Data_Label7 ]
local_df7 = pd.DataFrame(local_list7).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df7.index)], local_df7.values, palette='rainbow', alpha=.5)
plt.xticks(rotation = -45)
plt.xlabel('가해자법규위반')
plt.ylabel('사고 건수')
plt.show()

```





### 가해자 성별 & 피해자 성별

- 남
- 여
- 기타 불명
- 남자가 사고를 일으키는 경우가 압도적으로 많음.

```
In [50]: lbl8 = LabelEncoder()
Data_Label8 = lbl8.fit_transform( df['가해자성별'] )
Data_Label8
```

```
Out[50]: array([1, 1, 1, ..., 1, 1, 1])
```

```
In [51]: lbl9 = LabelEncoder()
Data_Label9 = lbl9.fit_transform( df['피해자성별'] )
Data_Label9
```

```
Out[51]: array([3, 1, 1, ..., 1, 1, 1])
```

```
In [52]: df_가해자성별_dic = { idx : sex for idx, sex in enumerate(list(lbl8.classes_)) }
df_가해자성별_dic
```

```
Out[52]: {0: '기타불명', 1: '남', 2: '여'}
```

```
In [53]: df_피해자성별_dic = { idx : sex for idx, sex in enumerate(list(lbl8.classes_)) }
df_피해자성별_dic
```

```
Out[53]: {0: '기타불명', 1: '남', 2: '여'}
```

```
In [54]: local_list8 = [ df_가해자성별_dic.get(i) for i in Data_Label8 ]
local_list9 = [ df_피해자성별_dic.get(i) for i in Data_Label9 ]

local_df8 = pd.DataFrame(local_list8).value_counts()
local_df9 = pd.DataFrame(local_list9).value_counts()

# sb.barplot([i[0] for i in list(local_df8.index)], local_df8.values, palette='Set2', alpha=.5)
# plt.xlabel('성별')
# plt.ylabel('사고 건수')

a=pd.DataFrame()
a['가해자성별']=local_df8
a['피해자성별']=local_df9

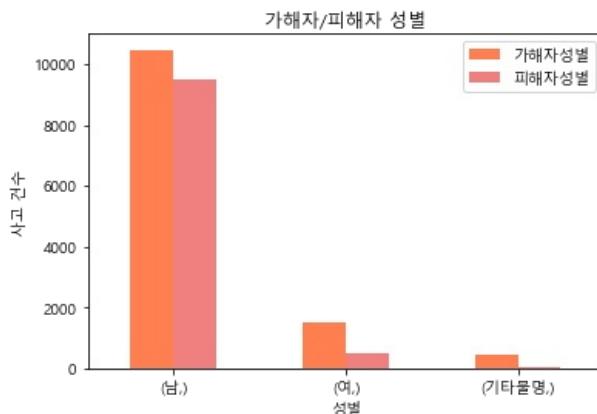
a=a.rename(index={'없음':'여'})

a.plot.bar(color=['coral', 'lightcoral'], rot=0)
```

```

plt.title('가해자/피해자 성별')
plt.xlabel('성별')
plt.ylabel('사고 건수')
plt.show()

```



```
In [55]: df['가해자성별'] = Data_Label8
df['피해자성별'] = Data_Label9
```

### 가해자연령

- 중장년층(40 ~ 60세)가 일으키는 사고 건수가 가장 많음.

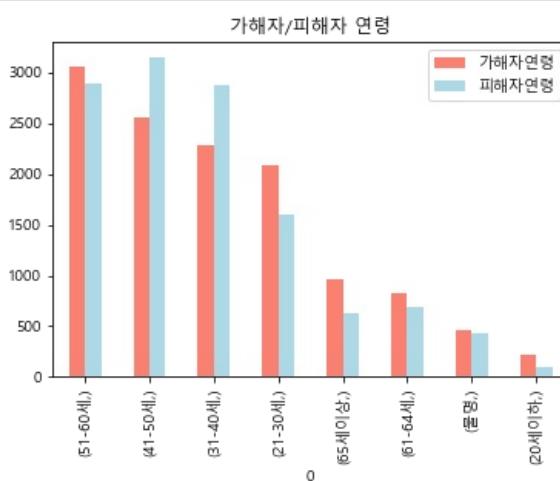
```
In [56]: lbl9 = LabelEncoder()
lbl10 = LabelEncoder()
Data_Label9 = lbl9.fit_transform(df['가해자연령'])
Data_Label10 = lbl10.fit_transform(df['피해자연령'])
```

```
In [57]: df_가해자연령_dic = { idx : age for idx, age in enumerate(list(lbl9.classes_)) }
df_피해자연령_dic = { idx : age for idx, age in enumerate(list(lbl10.classes_)) }
```

```
In [58]: local_list9 = [ df_가해자연령_dic.get(i) for i in Data_Label9 ]
local_list10 = [ df_피해자연령_dic.get(i) for i in Data_Label10 ]
local_df9 = pd.DataFrame(local_list9).value_counts()
local_df10 = pd.DataFrame(local_list10).value_counts()

a = pd.DataFrame()
a['가해자연령'] = local_df9
a['피해자연령'] = local_df10

a.plot.bar(color=['salmon','lightblue'])
plt.title('가해자/피해자 연령')
plt.show()
```



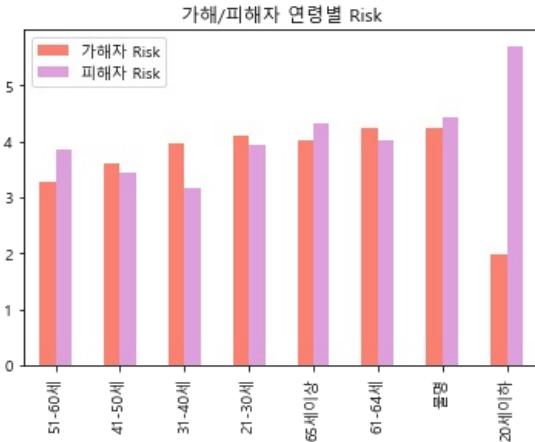
```
In [59]: s = [i[0] for i in a.index]
df_age_1 = df[['가해자연령','Risk']]
df_age_s1 = df_age_1.sort_values(by='가해자연령')
df_age_group1 = df_age_s1.groupby(by='가해자연령')
df_age_group1 = df_age_group1.mean()
```

```

df_age_2 = df[['피해자연령', 'Risk']]
df_age_s2 = df_age_2.sort_values(by='피해자연령')
df_age_group2 = df_age_s2.groupby(by='피해자연령')
df_age_group2 = df_age_group2.mean()[:-1]

a = pd.DataFrame()
a['연령'] = s
a.index = s
a['가해자 Risk']=df_age_group1['Risk'].values
a['피해자 Risk']=df_age_group2['Risk'].values
a.drop(['연령'], axis=1, inplace=True)
a.plot.bar(color = ['salmon', 'plum'])
plt.title('가해/피해자 연령별 Risk')
plt.show()

```



```
In [61]: df['가해자연령'] = Data_Label9
df['피해자연령'] = Data_Label10
```

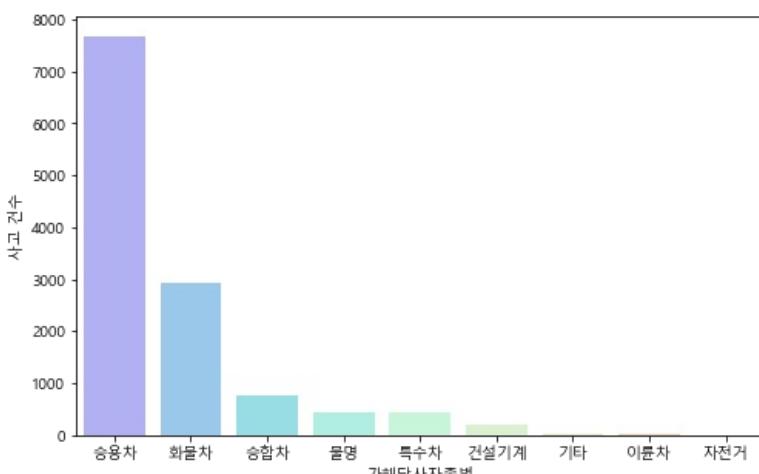
### 피해자 & 가해당사자종별

- 승용차, 화물차의 사고 빈도가 매우 높음
- 의외로 대형 사고 발생 가능성이 큰 화물차의 사고 빈도가 높다는 점에 주의

```
In [62]: lbl10 = LabelEncoder()
Data_Label10 = lbl10.fit_transform( df['가해당사자종별'] )
lbl11 = LabelEncoder()
Data_Label11 = lbl11.fit_transform( df['피해당사자종별'] )
```

```
In [63]: df_가해당사자종별_dic = { idx : age for idx, age in enumerate(list(lbl10.classes_)) }
df_피해당사자종별_dic = { idx : age for idx, age in enumerate(list(lbl11.classes_)) }
```

```
In [64]: local_list10 = [ df_가해당사자종별_dic.get(i) for i in Data_Label10 ]
local_list11 = [ df_피해당사자종별_dic.get(i) for i in Data_Label11 ]
local_df10 = pd.DataFrame(local_list10).value_counts()
local_df11 = pd.DataFrame(local_list11).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df10.index)], local_df10.values, palette='rainbow', alpha=.5)
plt.xlabel('가해당사자종별')
plt.ylabel('사고 건수')
plt.show()
```



```
In [65]: df['가해당사자종별'] = Data_Label10  
df['피해당사자종별'] = Data_Label11
```

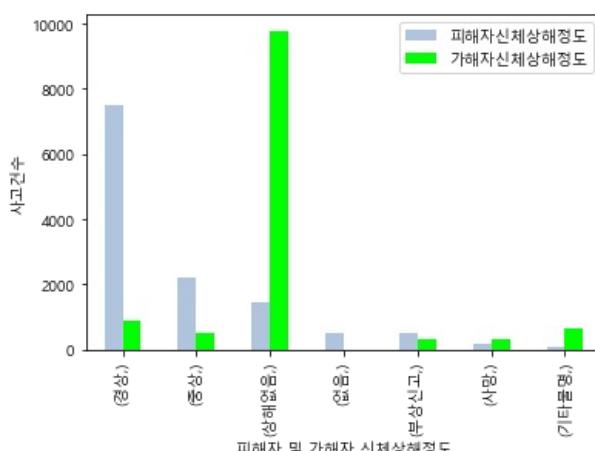
### 피해자신체상해정도 & 가해자신체상해정도

- 고속도로 위에서의 중상 비율이 매우 높음.

```
In [66]: lbl13 = LabelEncoder()  
lbl14 = LabelEncoder()  
Data_Label13 = lbl13.fit_transform(df['피해자신체상해정도'])  
Data_Label14 = lbl14.fit_transform(df['가해자신체상해정도'])
```

```
In [67]: df_피해자신체상해정도_dic = { idx : damage for idx, damage in enumerate(list(lbl13.classes_)) }  
df_가해자신체상해정도_dic = { idx : damage for idx, damage in enumerate(list(lbl14.classes_)) }
```

```
In [68]: local_list13 = [ df_피해자신체상해정도_dic.get(i) for i in Data_Label13 ]  
local_df13 = pd.DataFrame(local_list13).value_counts()  
  
local_list14 = [ df_가해자신체상해정도_dic.get(i) for i in Data_Label14 ]  
local_df14 = pd.DataFrame(local_list14).value_counts()  
  
# plt.figure(figsize=(8, 5))  
# sb.barplot([i[0] for i in list(local_df11.index)], local_df11.values, palette='rainbow', alpha=.5)  
# plt.xlabel('피해자신체상해정도')  
# plt.ylabel('사고 건수')  
  
신체상해정도비교_df = pd.DataFrame()  
신체상해정도비교_df['피해자신체상해정도'] = local_df13  
신체상해정도비교_df['가해자신체상해정도'] = local_df14  
  
신체상해정도비교_df.plot.bar(color=['lightsteelblue', 'lime'])  
plt.xlabel('피해자 및 가해자 신체상해정도')  
plt.ylabel('사고건수')  
plt.show()
```



```
In [69]: df['피해자신체상해정도'] = Data_Label13  
df['가해자신체상해정도'] = Data_Label14
```

### 가해차량용도\_대분류

- 비사업용 비율이 높은 것으로 보아, 일 도중 아닌 사고의 비율이 크다.

```
In [70]: lbl12 = LabelEncoder()  
Data_Label12 = lbl12.fit_transform(df['가해차량용도_대분류'])  
Data_Label12
```

```
Out[70]: array([2, 2, 3, ..., 2, 2, 2])
```

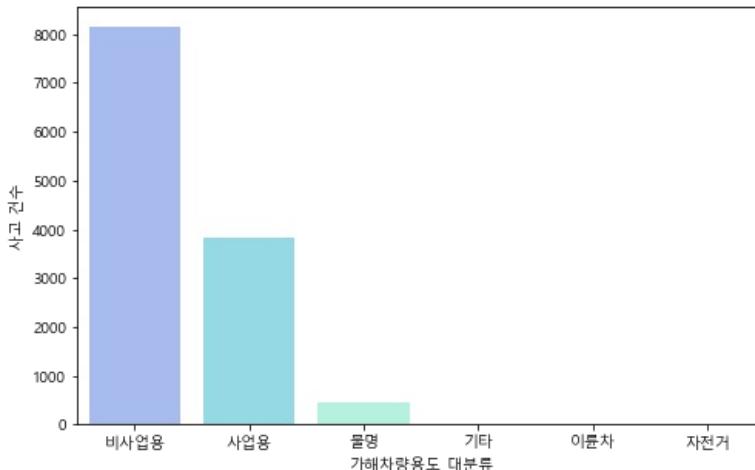
```
In [71]: df['가해차량용도_대분류'] = Data_Label12
```

```
In [72]: df['가해차량용도_대분류'] = Data_Label12
```

```
at_가해차량용도_내분류_dic = { idx : car for idx, car in enumerate(list(lbl12.classes_)) }
```

```
Out[72]: {0: '기타', 1: '불명', 2: '비사업용', 3: '사업용', 4: '이륜차', 5: '자전거'}
```

```
In [73]: local_list12 = [ df_가해차량용도_대분류_dic.get(i) for i in Data_Label12 ]
local_df12 = pd.DataFrame(local_list12).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df12.index)], local_df12.values, palette='rainbow', alpha=.5)
plt.xlabel('가해차량용도_대분류')
plt.ylabel('사고 건수')
plt.show()
```



### 가해차량용도\_중분류

- 승용차, 화물차의 사고 건수가 가장 많음.

```
In [74]: lbl13 = LabelEncoder()
Data_Label13 = lbl13.fit_transform( df['가해차량용도_중분류'] )
Data_Label13
```

```
Out[74]: array([7, 7, 9, ..., 7, 7, 7])
```

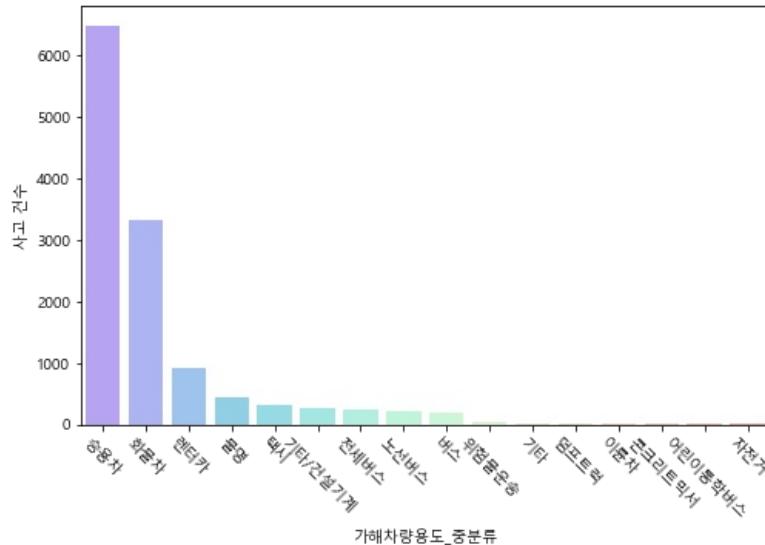
```
In [75]: df['가해차량용도_중분류'] = Data_Label13
```

```
In [76]: df_가해차량용도_중분류_dic = { idx : car for idx, car in enumerate(list(lbl13.classes_)) }
df_가해차량용도_중분류_dic
```

```
Out[76]: {0: '기타',
1: '기타/건설기계',
2: '노선버스',
3: '덤프트럭',
4: '렌터카',
5: '버스',
6: '불명',
7: '승용차',
8: '어린이통학버스',
9: '위험물운송',
10: '이륜차',
11: '자전거',
12: '전세버스',
13: '콘크리트믹서',
14: '택시',
15: '화물차'}
```

```
In [77]: local_list13 = [ df_가해차량용도_중분류_dic.get(i) for i in Data_Label13 ]
local_df13 = pd.DataFrame(local_list13).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df13.index)], local_df13.values, palette='rainbow', alpha=.5)
plt.xticks(rotation = -45)
plt.xlabel('가해차량용도_중분류')
```

```
plt.ylabel('사고 건수')
plt.show()
```



## 가해차량용도

```
In [78]: lbl14 = LabelEncoder()
Data_Label14 = lbl14.fit_transform( df['가해차량용도'] )
Data_Label14
```

```
Out[78]: array([ 9,  9, 13, ...,  9,  9,  9])
```

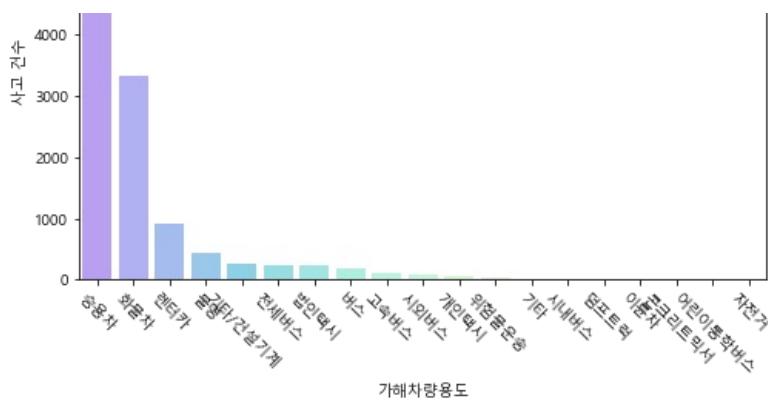
```
In [79]: df['가해차량용도'] = Data_Label14
```

```
In [80]: df_가해차량용도_dic = { idx : car for idx, car in enumerate(list(lbl14.classes_)) }
df_가해차량용도_dic
```

```
Out[80]: {0: '개인택시',
1: '고속버스',
2: '기타',
3: '기타/건설기계',
4: '덤프트럭',
5: '렌터카',
6: '버스',
7: '법인택시',
8: '불명',
9: '승용차',
10: '시내버스',
11: '시외버스',
12: '어린이통학버스',
13: '위험물운송',
14: '이륜차',
15: '자전거',
16: '전세버스',
17: '콘크리트믹서',
18: '화물차'}
```

```
In [81]: local_list14 = [ df_가해차량용도_dic.get(i) for i in Data_Label14 ]
local_df14 = pd.DataFrame(local_list14).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df14.index)], local_df14.values, palette='rainbow', alpha=.5)
plt.xticks(rotation = -45)
plt.xlabel('가해차량용도')
plt.ylabel('사고 건수')
plt.show()
```





## 피해차량용도\_대분류

```
In [82]: lbl15 = LabelEncoder()
Data_Label15 = lbl15.fit_transform( df['피해차량용도_대분류'] )
Data_Label15
```

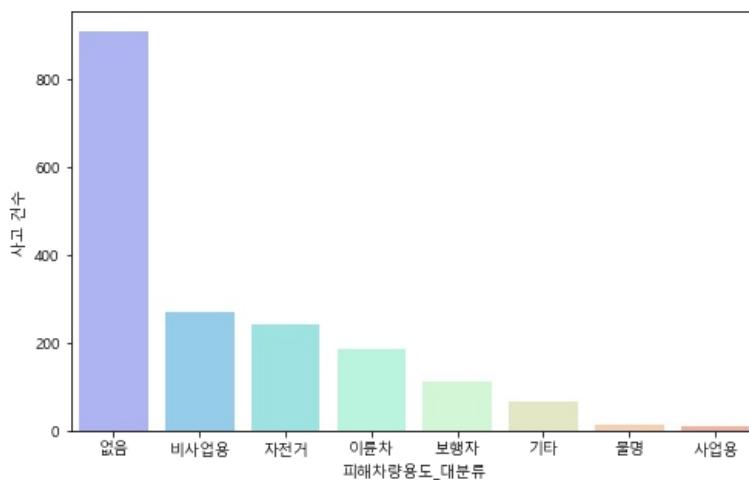
```
Out[82]: array([3, 3, 4, ..., 3, 4, 4])
```

```
In [83]: df['피해차량용도_대분류'] = Data_Label15
```

```
In [84]: df_피해차량용도_대분류_dic = { idx : car for idx, car in enumerate(list(lbl15.classes_)) }
df_피해차량용도_대분류_dic
```

```
Out[84]: {0: '기타', 1: '보행자', 2: '불명', 3: '비사업용', 4: '사업용', 5: '없음', 6: '이륜차', 7: '자전거'}
```

```
In [85]: local_list14 = [ df_피해차량용도_대분류_dic.get(i) for i in Data_Label14 ]
local_df14 = pd.DataFrame(local_list14).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df14.index)], local_df14.values, palette='rainbow', alpha=.5)
plt.xlabel('피해차량용도_대분류')
plt.ylabel('사고 건수')
plt.show()
```



## 피해차량용도\_중분류

```
In [86]: lbl16 = LabelEncoder()
Data_Label16 = lbl16.fit_transform( df['피해차량용도_중분류'] )
Data_Label16
```

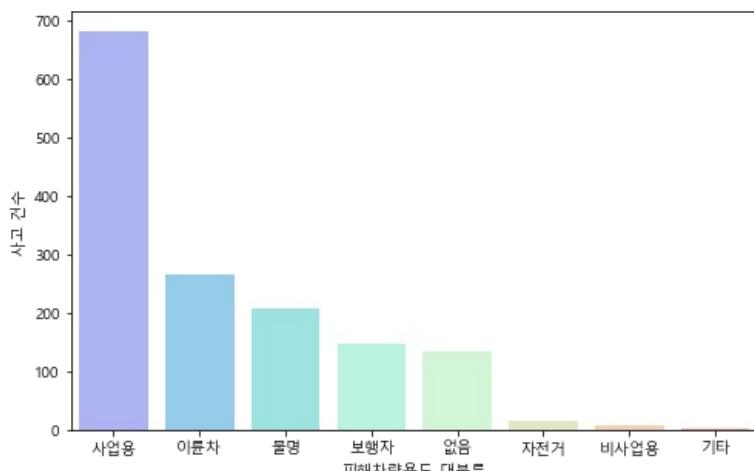
```
Out[86]: array([ 8,  8,  4, ...,  8, 14, 18])
```

```
In [87]: df['피해차량용도_중분류'] = Data_Label16
```

```
In [88]: df_피해차량용도_중분류_dic = { idx : car for idx, car in enumerate(list(lbl16.classes_)) }
```

```
Out[88]: {0: '기타',
 1: '기타/건설기계',
 2: '노선버스',
 3: '덤프트럭',
 4: '렌터카',
 5: '버스',
 6: '보행자',
 7: '불명',
 8: '승용차',
 9: '어린이통학버스',
 10: '없음',
 11: '위험물운송',
 12: '이륜차',
 13: '자전거',
 14: '전세버스',
 15: '콘크리트믹서',
 16: '택시',
 17: '특수여객(장의)',
 18: '화물차'}
```

```
In [89]: local_list16 = [ df_피해차량용도_대분류_dic.get(i) for i in Data_Label16 ]
local_df16 = pd.DataFrame(local_list16).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df16.index)], local_df16.values, palette='rainbow', alpha=.5)
plt.xlabel('피해차량용도_대분류')
plt.ylabel('사고 건수')
plt.show()
```



## 피해차량용도

```
In [90]: lbl17 = LabelEncoder()
Data_Label17 = lbl17.fit_transform( df['피해차량용도'] )
Data_Label17
```

```
Out[90]: array([11, 11, 5, ..., 11, 19, 22])
```

```
In [91]: df['피해차량용도'] = Data_Label17
```

```
In [92]: df_피해차량용도_dic = { idx : car for idx, car in enumerate(list(lbl17.classes_)) }
df_피해차량용도_dic
```

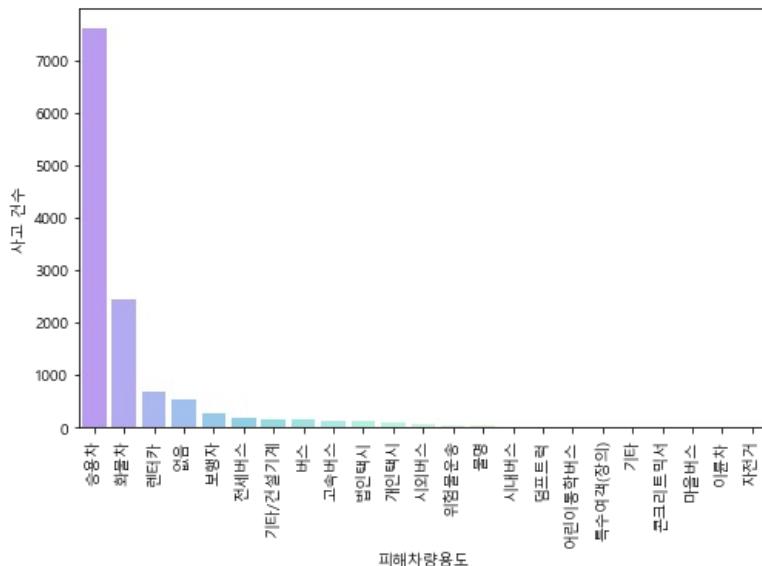
```
Out[92]: {0: '개인택시',
 1: '고속버스',
 2: '기타',
 3: '기타/건설기계',
 4: '덤프트럭',
 5: '렌터카',
 6: '마을버스',
```

```

7: '버스',
8: '법인택시',
9: '보행자',
10: '불명',
11: '승용차',
12: '시내버스',
13: '시외버스',
14: '어린이통학버스',
15: '없음',
16: '위험물운송',
17: '이륜차',
18: '자전거',
19: '전세버스',
20: '콘크리트믹서',
21: '특수여객(장의)',
22: '화물차'}

```

```
In [93]: local_list17 = [ df_피해차량용도_dict.get(i) for i in Data_Label17 ]
local_df17 = pd.DataFrame(local_list17).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df17.index)], local_df17.values, palette='rainbow', alpha=.5)
plt.xticks(rotation = 90)
plt.xlabel('피해차량용도')
plt.ylabel('사고 건수')
plt.show()
```



## 기상상태 & 노면상태

```
In [94]: lbl22 = LabelEncoder()
Data_Label22 = lbl22.fit_transform( df['노면상태'] )
df_노면상태_dict = { idx : car for idx, car in enumerate(list(lbl22.classes_)) }

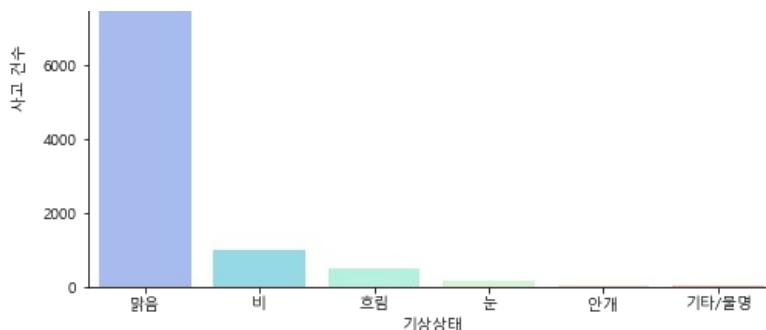
lbl23 = LabelEncoder()
Data_Label23 = lbl23.fit_transform( df['기상상태'] )
df_기상상태_dict = { idx : car for idx, car in enumerate(list(lbl23.classes_)) }

df['노면상태'] = Data_Label22
local_list22 = [ df_노면상태_dict.get(i) for i in Data_Label22 ]

df['기상상태'] = Data_Label23
local_list23 = [ df_기상상태_dict.get(i) for i in Data_Label23 ]
```

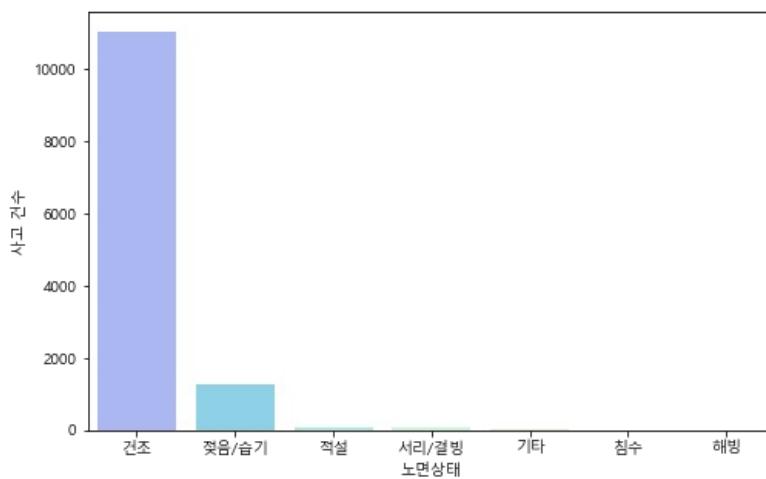
```
In [95]: local_df23 = pd.DataFrame(local_list23).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df23.index)], local_df23.values, palette='rainbow', alpha=.5)
plt.xlabel('기상상태')
plt.ylabel('사고 건수')
plt.show()
```





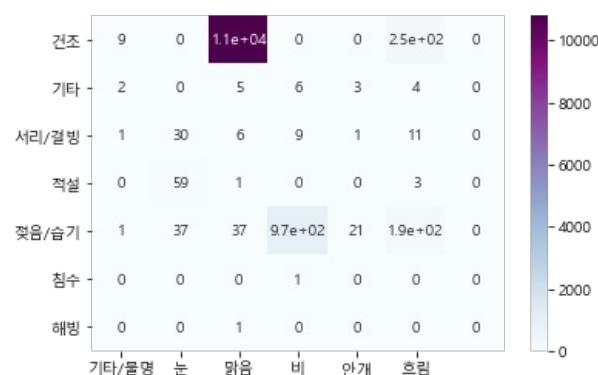
In [96]:

```
local_list22 = [ df_노면상태_dic.get(i) for i in Data_Label22 ]
local_df22 = pd.DataFrame(local_list22).value_counts()
plt.figure(figsize=(8, 5))
sb.barplot([i[0] for i in list(local_df22.index)], local_df22.values, palette='rainbow', alpha=.5)
plt.xlabel('노면상태')
plt.ylabel('사고 건수')
plt.show()
```



In [97]:

```
import sklearn.metrics as metrics
df1 = df[['노면상태', '기상상태']]
cm = metrics.confusion_matrix(df['노면상태'].values, df['기상상태'].values)
sb.heatmap(cm, annot=True, cmap='BuPu', xticklabels=list(lbl23.classes_), yticklabels=list(lbl22.classes_))
plt.show()
```



- <https://blog.naver.com/chunsa0127/222074830394>

In [98]:

```
def lambda_coefficient(data) :
    max_column = np.max(data.sum(axis=0))
    denominator = np.sum(data.sum()) - max_column
    numerator = data.apply(lambda x: np.sum(x)- np.max(x), axis=1)
    numerator = np.sum(numerator)
    return round(numerator/denominator, 4)
```

In [99]:

```
print(f'기상상태와 노면상태의 람다계수 : {lambda_coefficient(df1)}')
```

기상상태와 노면상태의 람다계수 : 0.7548

## Feature간의 람다 상관 계수값 파악

- 75% 이하의 조합만을 추출(상대적으로 상관관계가 낮게 판단되는 케이스)
- 발생지\_시도 변수가 다른 변수와의 상관성이 상대적으로 매우 낮은 양상을 보임.

```
In [100]:  
# for i , j in list(combinations(df.iloc[:, 1: 25].columns, 2)) :  
#     data = df[[i, j]]  
#     coef = lambda coefficient(data)  
#     if coef < 0.75 :  
#         print(f'{i}와 {j}의 람다 상관 계수값 : {coef}')
```

데이터 프레임 전처리 정리

```
In [101]:  
# df.to_csv('data1.csv', encoding = 'euc-kr')
```

지역명 반환 함수

```
In [100]:  
df1 = df.copy()  
local_list = [df_발생지_시군구_dic.get(i) for i in df['발생지_시군구']]  
df1['발생지_시군구'] = local_list  
df2 = df1.groupby('발생지_시군구').mean()['Risk']  
df2.to_csv('map.csv')
```

```
In [ ]:  
df1 = df.copy()  
local_list = [df_발생지_시군구_dic.get(i) for i in df['발생지_시군구']]  
df1['발생지_시군구'] = local_list  
df2 = df1.groupby('발생지_시군구').mean()['응급실 개수']  
df2.to_csv('emergency.csv')
```

```
In [ ]:  
df1 = df.copy()  
local_list = [df_발생지_시군구_dic.get(i) for i in df['발생지_시군구']]  
df1['발생지_시군구'] = local_list  
df2 = df1.groupby('발생지_시군구').mean()['시군구별_인구']  
df2.to_csv('pop.csv')
```

Loading [MathJax]/extensions/Safe.js

In [1]:

```

import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
%matplotlib inline
from tqdm.notebook import tqdm
from collections import Counter
from itertools import combinations
from IPython.display import Image

import scipy.stats
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf

from scipy.stats import shapiro
from statsmodels.stats.proportion import proportions_ztest
from statsmodels.stats.outliers_influence import variance_inflation_factor

import sklearn.metrics as metrics
from sklearn.feature_selection import RFE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import KFold, cross_validate, cross_val_score

from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet

plt.rcParams['font.family'] = 'Malgun Gothic'
plt.rc('axes', unicode_minus=False)
plt.rcParams['font.size'] = 16
plt.rcParams['figure.figsize'] = 8, 5
plt.rcParams['axes.unicode_minus'] = False

import warnings
warnings.filterwarnings('ignore')

```

In [2]:

```

df = pd.read_csv('data1.csv', encoding = 'euc-kr' , index_col = 'Unnamed: 0' )
df.drop(['발생일'], axis = 1, inplace = True)

```

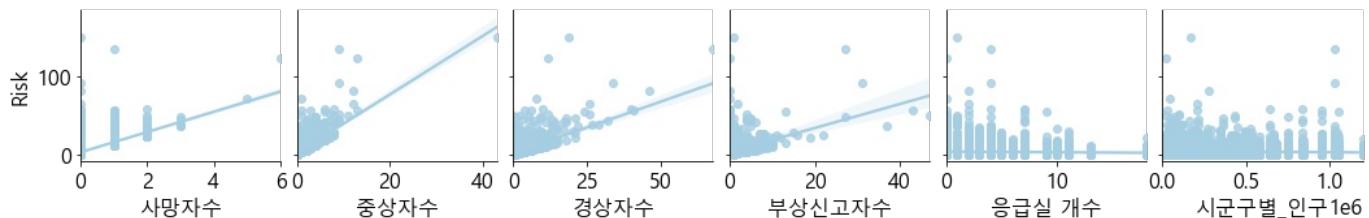
## 연속형 변수에 대한 EDA

In [3]:

```

sb.set_palette('Paired')
sb.pairplot(df, y_vars = 'Risk', x_vars = list(df.columns)[-7:-1], kind="reg")
plt.show()

```



## 상관분석

- Step 1. Pearson 상관계수 및 Heatmap을 통한 상관분석 실시

- 변수(feature)들의 상관관계를 고려하기 위해 DataFrame.corr() 함수 : default = Pearson을 이용하여 heatmap으로 표시한 결과, 사망자수, 중상자수, 경상자수, 부상신고자수 feature가 Risk(Target)에 영향을 가장 크게 끼쳤다.
- Step1의 분석 및 문제
  - Risk 계산시 위 4가지 변수에 대한 가중치 값의 합을 이용하였으므로, 예상된 결과라고 생각되며, 다중공선성 문제를 야기시킬 수

있는 변수로 모델링시 제거할 필요가 있다.

- 명목변수 데이터 간의 상관분석은 Pearson 상관계수로는 올바른 관계 해석이 이루어질 수 없으므로, 명목 변수간의 관계를 파악 할 수 있는 lambda 계수를 추가적으로 활용할 필요가 있다.

#### • Step 2. lambda 계수를 통한 상관분석 실시

- lambda 계수의 정의 : 명목변수들 간 상관관계의 크기를 나타내는 계수. 일명 거트만(Guttman) 상관계수.

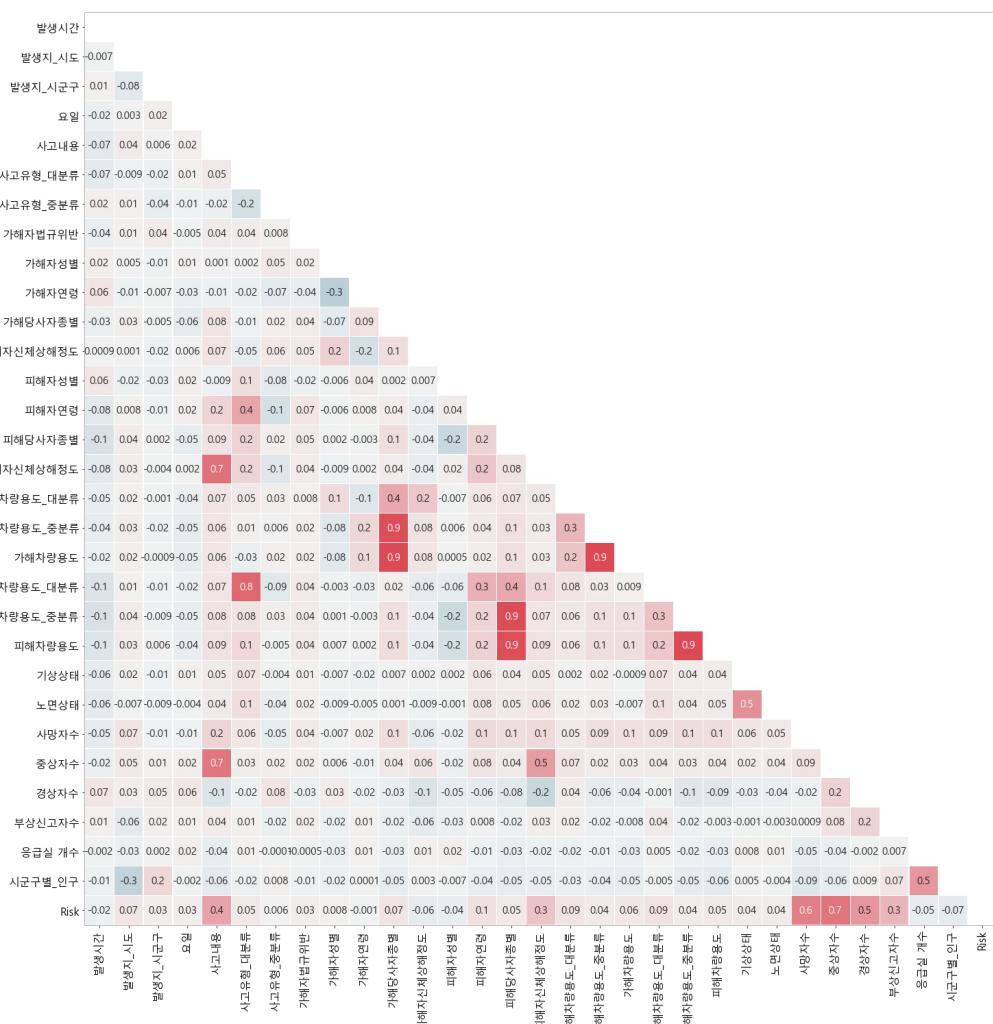
#### Step 1. Pearson 상관계수 및 Heatmap을 통한 상관분석 실시

In [4]:

```
f, ax = plt.subplots(figsize=(35,35))
matrix = np.triu(df.corr())

plt.title("Correlation of Features", y = 1.05, size = 40)
colormap = sb.diverging_palette(220, 10, as_cmap = True)
sb.heatmap(df.corr(),
            annot = True, annot_kws = {"size": 15}, fmt = '.1g', linewidths = 0.1,
            center = 0, vmin = -1, vmax = 1, linecolor = "white",
            cmap = colormap, cbar = True, cbar_kws = {'orientation': 'horizontal'},
            square = True, mask = matrix)
plt.show()
```

Correlation of Features



#### Step 2. lambda 계수를 통한 상관분석 실시

- 명목 변수간의 다중공선성을 일으킬 수 있을 만한 요인을 탐색.
- 두 명목변수 간의 쌍에 대한 상관관계가 0.8이상인 모든 조합의 수를 탐색한 후, 가장 많이 등장한 변수명에 대한 빈도 시각화 수행.
- 명목변수 간의 상관관계가 높아 다중공선성을 일으킬 수 있는 변수 목록

- [발생시간, 발생시\_시군구, 사고유형\_대분류, 가해자법규위반, 가해자성별, 가해 당사자 종별, 피해자 성별, 피해당사자종별, 가해자차량용도\_중분류, 가해자차량용도\_대분류, 피해자차량용도\_대분류, 기상상태. 가해자차량용도, 피해자차량용도, 피해자차량용도\_중분류]

In [5]:

```
def lambda_coefficient(data) :
    max_column = np.max(data.sum(axis=0))
    denominator = np.sum(data.sum()) - max_column
    numerator = data.apply(lambda x: np.sum(x) - np.max(x), axis=1)
    numerator = np.sum(numerator)
    return round(numerator/denominator, 4)
```

In [6]:

```
low_coef = []
for i, j in list(combinations(df.iloc[:, :-7].columns, 2)) :
    data = df[[i, j]]
    coef = lambda_coefficient(data)
    if coef > 0.8 :
        print(f'{i}와 {j}의 람다 상관 계수값 : {coef}')
        s = i + '와 ' + j
        low_coef.append(s)
```

발생시간과 발생지\_시도의 람다 상관 계수값 : 0.8271  
 발생시간과 발생지\_시군구의 람다 상관 계수값 : 0.9634  
 발생시간과 요일의 람다 상관 계수값 : 0.9308  
 발생시간과 사고내용의 람다 상관 계수값 : 0.9349  
 발생시간과 사고유형\_대분류의 람다 상관 계수값 : 0.9678  
 발생시간과 사고유형\_중분류의 람다 상관 계수값 : 0.8362  
 발생시간과 가해자법규위반의 람다 상관 계수값 : 0.9235  
 발생시간과 가해자성별의 람다 상관 계수값 : 0.97  
 발생시간과 가해자연령의 람다 상관 계수값 : 0.9438  
 발생시간과 가해당사자종별의 람다 상관 계수값 : 0.9187  
 발생시간과 가해자신체상해정도의 람다 상관 계수값 : 0.9358  
 발생시간과 피해자성별의 람다 상관 계수값 : 0.9655  
 발생시간과 피해자연령의 람다 상관 계수값 : 0.9246  
 발생시간과 피해당사자종별의 람다 상관 계수값 : 0.8751  
 발생시간과 피해자신체상해정도의 람다 상관 계수값 : 0.8969  
 발생시간과 가해차량용도\_대분류의 람다 상관 계수값 : 0.9509  
 발생시간과 가해차량용도\_중분류의 람다 상관 계수값 : 0.818  
 발생시간과 피해차량용도\_대분류의 람다 상관 계수값 : 0.9333  
 발생시간과 기상상태의 람다 상관 계수값 : 0.9494  
 발생시간과 노면상태의 람다 상관 계수값 : 0.9007  
 발생지\_시도와 발생지\_시군구의 람다 상관 계수값 : 0.9785  
 발생지\_시도와 사고유형\_대분류의 람다 상관 계수값 : 0.9399  
 발생지\_시도와 가해자성별의 람다 상관 계수값 : 0.9012  
 발생지\_시도와 피해자성별의 람다 상관 계수값 : 0.8069  
 발생지\_시도와 가해차량용도의 람다 상관 계수값 : 0.8067  
 발생지\_시도와 피해차량용도의 람다 상관 계수값 : 0.8759  
 발생지\_시군구와 요일의 람다 상관 계수값 : 0.9899  
 발생지\_시군구와 사고내용의 람다 상관 계수값 : 0.9949  
 발생지\_시군구와 사고유형\_대분류의 람다 상관 계수값 : 0.9972  
 발생지\_시군구와 사고유형\_중분류의 람다 상관 계수값 : 0.9735  
 발생지\_시군구와 가해자법규위반의 람다 상관 계수값 : 0.989  
 발생지\_시군구와 가해자성별의 람다 상관 계수값 : 0.9969  
 발생지\_시군구와 가해자연령의 람다 상관 계수값 : 0.9907  
 발생지\_시군구와 가해당사자종별의 람다 상관 계수값 : 0.9876  
 발생지\_시군구와 가해자신체상해정도의 람다 상관 계수값 : 0.9915  
 발생지\_시군구와 피해자성별의 람다 상관 계수값 : 0.9959  
 발생지\_시군구와 피해자연령의 람다 상관 계수값 : 0.9915  
 발생지\_시군구와 피해당사자종별의 람다 상관 계수값 : 0.9872  
 발생지\_시군구와 피해자신체상해정도의 람다 상관 계수값 : 0.9879  
 발생지\_시군구와 가해차량용도\_대분류의 람다 상관 계수값 : 0.9958  
 발생지\_시군구와 가해차량용도\_중분류의 람다 상관 계수값 : 0.9739  
 발생지\_시군구와 가해차량용도의 람다 상관 계수값 : 0.9698  
 발생지\_시군구와 피해차량용도\_대분류의 람다 상관 계수값 : 0.9934  
 발생지\_시군구와 피해차량용도\_중분류의 람다 상관 계수값 : 0.973  
 발생지\_시군구와 피해차량용도의 람다 상관 계수값 : 0.9661  
 발생지\_시군구와 기상상태의 람다 상관 계수값 : 0.9962  
 발생지\_시군구와 노면상태의 람다 상관 계수값 : 0.9935  
 요일과 사고유형\_대분류의 람다 상관 계수값 : 0.834  
 요일과 가해자법규위반의 람다 상관 계수값 : 0.8945  
 요일과 가해자성별의 람다 상관 계수값 : 0.8241  
 요일과 피해당사자종별의 람다 상관 계수값 : 0.8816  
 요일과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9641  
 요일과 가해차량용도의 람다 상관 계수값 : 0.9781  
 요일과 피해차량용도\_중분류의 람다 상관 계수값 : 0.974  
 요일과 피해차량용도의 람다 상관 계수값 : 0.9784  
 사고내용과 사고유형\_중분류의 람다 상관 계수값 : 0.8049  
 사고내용과 가해자법규위반의 람다 상관 계수값 : 0.9665  
 사고내용과 가해자연령의 람다 상관 계수값 : 0.828  
 사고내용과 가해당사자종별의 람다 상관 계수값 : 0.9777  
 사고내용과 가해자신체상해정도의 람다 상관 계수값 : 0.911  
 사고내용과 피해자연령의 람다 상관 계수값 : 0.8759

사고내용과 피해당사자종별의 람다 상관 계수값 : 0.9823  
사고내용과 가해차량용도\_대분류의 람다 상관 계수값 : 0.8042  
사고내용과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9778  
사고내용과 가해차량용도의 람다 상관 계수값 : 0.9858  
사고내용과 피해차량용도\_대분류의 람다 상관 계수값 : 0.9771  
사고내용과 피해차량용도\_중분류의 람다 상관 계수값 : 0.9861  
사고내용과 피해차량용도의 람다 상관 계수값 : 0.9857  
사고유형\_대분류와 사고유형\_중분류의 람다 상관 계수값 : 0.9468  
사고유형\_대분류와 가해자법규위반의 람다 상관 계수값 : 0.9887  
사고유형\_대분류와 가해자성별의 람다 상관 계수값 : 0.9265  
사고유형\_대분류와 가해자연령의 람다 상관 계수값 : 0.9742  
사고유형\_대분류와 가해당사자종별의 람다 상관 계수값 : 0.9849  
사고유형\_대분류와 가해자신체상해정도의 람다 상관 계수값 : 0.9264  
사고유형\_대분류와 피해자성별의 람다 상관 계수값 : 0.9972  
사고유형\_대분류와 피해자연령의 람다 상관 계수값 : 0.9936  
사고유형\_대분류와 피해당사자종별의 람다 상관 계수값 : 0.993  
사고유형\_대분류와 가해차량용도\_대분류의 람다 상관 계수값 : 0.9988  
사고유형\_대분류와 가해차량용도\_중분류의 람다 상관 계수값 : 0.9976  
사고유형\_대분류와 가해차량용도의 람다 상관 계수값 : 0.9938  
사고유형\_대분류와 피해차량용도\_대분류의 람다 상관 계수값 : 0.9998  
사고유형\_대분류와 피해차량용도\_중분류의 람다 상관 계수값 : 0.9998  
사고유형\_대분류와 피해차량용도의 람다 상관 계수값 : 0.9938  
사고유형\_대분류와 기상상태의 람다 상관 계수값 : 0.9985  
사고유형\_중분류와 가해자성별의 람다 상관 계수값 : 0.9475  
사고유형\_중분류와 피해자성별의 람다 상관 계수값 : 0.8779  
사고유형\_중분류와 가해차량용도\_대분류의 람다 상관 계수값 : 0.8277  
사고유형\_중분류와 가해차량용도의 람다 상관 계수값 : 0.8056  
사고유형\_중분류와 피해차량용도의 람다 상관 계수값 : 0.9113  
사고유형\_중분류와 기상상태의 람다 상관 계수값 : 0.8297  
가해자법규위반과 가해자성별의 람다 상관 계수값 : 0.9881  
가해자법규위반과 가해자연령의 람다 상관 계수값 : 0.9009  
가해자법규위반과 가해자신체상해정도의 람다 상관 계수값 : 0.96  
가해자법규위반과 피해자성별의 람다 상관 계수값 : 0.979  
가해자법규위반과 피해자연령의 람다 상관 계수값 : 0.9143  
가해자법규위반과 피해당사자종별의 람다 상관 계수값 : 0.8796  
가해자법규위반과 피해자신체상해정도의 람다 상관 계수값 : 0.8439  
가해자법규위반과 가해차량용도\_대분류의 람다 상관 계수값 : 0.9797  
가해자법규위반과 가해차량용도\_중분류의 람다 상관 계수값 : 0.962  
가해자법규위반과 가해차량용도의 람다 상관 계수값 : 0.9798  
가해자법규위반과 피해차량용도\_대분류의 람다 상관 계수값 : 0.9667  
가해자법규위반과 피해차량용도\_중분류의 람다 상관 계수값 : 0.9731  
가해자법규위반과 피해차량용도의 람다 상관 계수값 : 0.9811  
가해자법규위반과 기상상태의 람다 상관 계수값 : 0.9794  
가해자법규위반과 노면상태의 람다 상관 계수값 : 0.9534  
가해자성별과 가해자연령의 람다 상관 계수값 : 0.9597  
가해자성별과 가해당사자종별의 람다 상관 계수값 : 0.9861  
가해자성별과 가해자신체상해정도의 람다 상관 계수값 : 0.9203  
가해자성별과 피해자성별의 람다 상관 계수값 : 0.9128  
가해자성별과 피해자연령의 람다 상관 계수값 : 0.9781  
가해자성별과 피해당사자종별의 람다 상관 계수값 : 0.9927  
가해자성별과 가해차량용도\_대분류의 람다 상관 계수값 : 0.999  
가해자성별과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9985  
가해자성별과 가해차량용도의 람다 상관 계수값 : 0.9972  
가해자성별과 피해차량용도\_대분류의 람다 상관 계수값 : 0.9982  
가해자성별과 피해차량용도\_중분류의 람다 상관 계수값 : 0.9986  
가해자성별과 피해차량용도의 람다 상관 계수값 : 0.9983  
가해자성별과 기상상태의 람다 상관 계수값 : 0.9983  
가해자연령과 가해당사자종별의 람다 상관 계수값 : 0.818  
가해자연령과 피해자성별의 람다 상관 계수값 : 0.9143  
가해자연령과 피해당사자종별의 람다 상관 계수값 : 0.9055  
가해자연령과 가해차량용도\_대분류의 람다 상관 계수값 : 0.8751  
가해자연령과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9604  
가해자연령과 가해차량용도의 람다 상관 계수값 : 0.9782  
가해자연령과 피해차량용도\_중분류의 람다 상관 계수값 : 0.9774  
가해자연령과 피해차량용도의 람다 상관 계수값 : 0.9812  
가해자연령과 기상상태의 람다 상관 계수값 : 0.8621  
가해당사자종별과 가해자신체상해정도의 람다 상관 계수값 : 0.827  
가해당사자종별과 피해자성별의 람다 상관 계수값 : 0.9773  
가해당사자종별과 피해자연령의 람다 상관 계수값 : 0.8353  
가해당사자종별과 피해당사자종별의 람다 상관 계수값 : 0.8181  
가해당사자종별과 가해차량용도\_대분류의 람다 상관 계수값 : 0.9817  
가해당사자종별과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9862  
가해당사자종별과 가해차량용도의 람다 상관 계수값 : 0.9875  
가해당사자종별과 피해차량용도\_대분류의 람다 상관 계수값 : 0.9146  
가해당사자종별과 피해차량용도\_중분류의 람다 상관 계수값 : 0.965  
가해당사자종별과 피해차량용도의 람다 상관 계수값 : 0.9731  
가해당사자종별과 기상상태의 람다 상관 계수값 : 0.9592  
가해당사자종별과 노면상태의 람다 상관 계수값 : 0.8298  
가해자신체상해정도와 피해자성별의 람다 상관 계수값 : 0.9131  
가해자신체상해정도와 피해당사자종별의 람다 상관 계수값 : 0.9769  
가해자신체상해정도와 가해차량용도\_대분류의 람다 상관 계수값 : 0.9199  
가해자신체상해정도와 가해차량용도\_중분류의 람다 상관 계수값 : 0.9714  
가해자신체상해정도와 가해차량용도의 람다 상관 계수값 : 0.9806

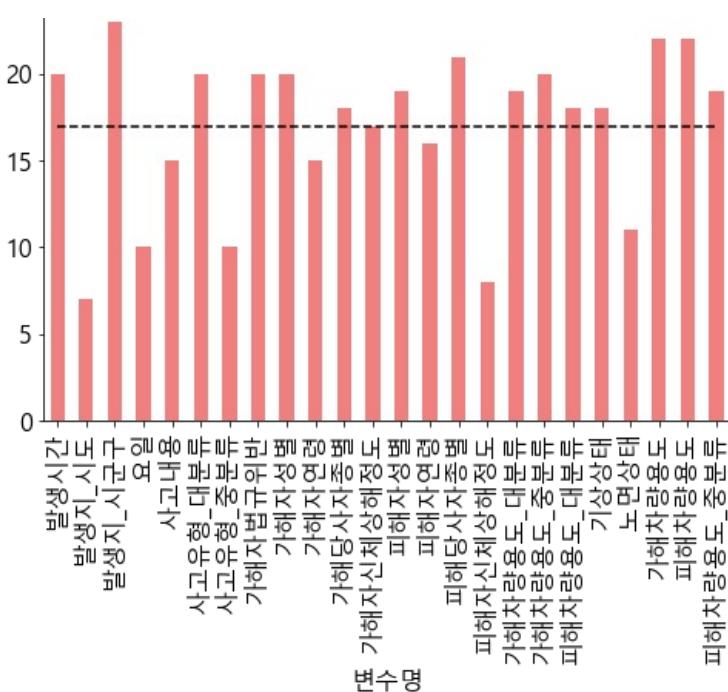
가해자신체상해정도와 피해차량용도\_대분류의 람다 상관 계수값 : 0.8664  
 가해자신체상해정도와 피해차량용도\_중분류의 람다 상관 계수값 : 0.9824  
 가해자신체상해정도와 피해차량용도의 람다 상관 계수값 : 0.9829  
 가해자신체상해정도와 기상상태의 람다 상관 계수값 : 0.8846  
 가해자신체상해정도와 노면상태의 람다 상관 계수값 : 0.8617  
 피해자성별과 피해자연령의 람다 상관 계수값 : 0.9225  
 피해자성별과 피해당사자종별의 람다 상관 계수값 : 0.9904  
 피해자성별과 가해차량용도\_대분류의 람다 상관 계수값 : 0.8973  
 피해자성별과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9901  
 피해자성별과 가해차량용도의 람다 상관 계수값 : 0.9915  
 피해자성별과 피해차량용도 대분류의 람다 상관 계수값 : 0.9909  
 피해자성별과 피해차량용도 중분류의 람다 상관 계수값 : 0.9987  
 피해자성별과 피해차량용도의 람다 상관 계수값 : 0.9952  
 피해자성별과 기상상태의 람다 상관 계수값 : 0.8759  
 피해자연령과 피해당사자종별의 람다 상관 계수값 : 0.9333  
 피해자연령과 가해차량용도\_대분류의 람다 상관 계수값 : 0.895  
 피해자연령과 가해차량용도\_중분류의 람다 상관 계수값 : 0.9652  
 피해자연령과 가해차량용도의 람다 상관 계수값 : 0.9768  
 피해자연령과 피해차량용도\_대분류의 람다 상관 계수값 : 0.8315  
 피해자연령과 피해차량용도\_중분류의 람다 상관 계수값 : 0.9812  
 피해자연령과 피해차량용도의 람다 상관 계수값 : 0.9798  
 피해자연령과 기상상태의 람다 상관 계수값 : 0.8936  
 피해당사자종별과 피해자신체상해정도의 람다 상관 계수값 : 0.8373  
 피해당사자종별과 가해차량용도\_대분류의 람다 상관 계수값 : 0.9894  
 피해당사자종별과 가해차량용도\_중분류의 람다 상관 계수값 : 0.8992  
 피해당사자종별과 가해차량용도의 람다 상관 계수값 : 0.9462  
 피해당사자종별과 피해차량용도\_대분류의 람다 상관 계수값 : 0.9918  
 피해당사자종별과 피해차량용도\_중분류의 람다 상관 계수값 : 0.9857  
 피해당사자종별과 피해차량용도의 람다 상관 계수값 : 0.9848  
 피해당사자종별과 기상상태의 람다 상관 계수값 : 0.9804  
 피해당사자종별과 노면상태의 람다 상관 계수값 : 0.9876  
 피해자신체상해정도와 가해차량용도\_중분류의 람다 상관 계수값 : 0.9511  
 피해자신체상해정도와 가해차량용도의 람다 상관 계수값 : 0.9683  
 피해자신체상해정도와 피해차량용도\_중분류의 람다 상관 계수값 : 0.9753  
 가해차량용도\_대분류와 가해차량용도\_중분류의 람다 상관 계수값 : 0.9781  
 가해차량용도\_대분류와 가해차량용도의 람다 상관 계수값 : 0.985  
 가해차량용도\_대분류와 피해차량용도\_대분류의 람다 상관 계수값 : 0.9877  
 가해차량용도\_대분류와 피해차량용도\_중분류의 람다 상관 계수값 : 0.9894  
 가해차량용도\_대분류와 피해차량용도의 람다 상관 계수값 : 0.9877  
 가해차량용도\_대분류와 기상상태의 람다 상관 계수값 : 0.9105  
 가해차량용도\_중분류와 가해차량용도의 람다 상관 계수값 : 0.9752  
 가해차량용도\_중분류와 피해차량용도\_대분류의 람다 상관 계수값 : 0.9751  
 가해차량용도\_중분류와 피해차량용도의 람다 상관 계수값 : 0.8721  
 가해차량용도\_중분류와 기상상태의 람다 상관 계수값 : 0.9849  
 가해차량용도\_중분류와 노면상태의 람다 상관 계수값 : 0.9787  
 가해차량용도와 피해차량용도\_대분류의 람다 상관 계수값 : 0.9857  
 가해차량용도와 피해차량용도\_중분류의 람다 상관 계수값 : 0.8221  
 가해차량용도와 피해차량용도의 람다 상관 계수값 : 0.8317  
 가해차량용도와 기상상태의 람다 상관 계수값 : 0.9889  
 가해차량용도와 노면상태의 람다 상관 계수값 : 0.982  
 피해차량용도\_대분류와 피해차량용도\_중분류의 람다 상관 계수값 : 0.9806  
 피해차량용도\_대분류와 피해차량용도의 람다 상관 계수값 : 0.9812  
 피해차량용도\_대분류와 기상상태의 람다 상관 계수값 : 0.9612  
 피해차량용도\_대분류와 노면상태의 람다 상관 계수값 : 0.8395  
 피해차량용도\_중분류와 피해차량용도의 람다 상관 계수값 : 0.9803  
 피해차량용도\_중분류와 기상상태의 람다 상관 계수값 : 0.9914  
 피해차량용도\_중분류와 노면상태의 람다 상관 계수값 : 0.9846  
 피해차량용도와 기상상태의 람다 상관 계수값 : 0.9877  
 피해차량용도와 노면상태의 람다 상관 계수값 : 0.9846

In [7]:

```
variable_dic = []
for i in range(len(low_coef)) :
    variable_dic += [low_coef[i].split('와')[0].strip()]
    variable_dic += [low_coef[i].split('와')[-1].strip()]

variable = []
count = []
for i, j in Counter(variable_dic).items() :
    variable.append(i)
    count.append(j)

variable_df = pd.DataFrame()
variable_df['변수명'] = variable
variable_df['빈도'] = count
variable_df.index = variable_df['변수명']
variable_df.drop(['변수명'], axis = 1, inplace = True)
bar_list = variable_df.plot.bar(y = "빈도", legend = False, color = 'lightcoral')
plt.plot([0, 23], [variable_df.mean(), variable_df.mean()], 'k--')
plt.show()
```



## 통계적 모델링(OLS)

- 종속변수(Risk : 연속형 데이터) ~ 설명변수(모든 변수 포함) : 연속형 데이터 + 범주형 데이터
    - LSE 방법론을 활용한 다중 회귀 분석 실시
    - 변수 선택 방법론으로서 후진제거법 이용.
    - Full model로부터 시작.
  - 분석 결과 및 해석
    - 결정계수(R-squared) : 0.217(약 22%)의 설명력을 가짐.
    - 수정된 결정계수(Adj. R-squared) : 0.216(약 22%)의 설명력을 가짐.
    - 모델에 대한 가설 검정
      - F-statistic : 132.8
      - P-value(F-statistic) : 0.00
      - 유의수준 0.05에서 P값이 충분히 작으므로 모델은 유의미하다고 볼 수 있다.
      - 그러나 모델에 대한 설명력이 매우 낮아 사용하기는 어려울 것으로 판단된다.
    - 유의수준 5%에서 유의미하지 않은 변수.
      - [가해자법규위반, 가해자연령, 기상상태, 노면상태, 응급실 개수]

```
In [8]: # OLS 과정에서 변수 하나를 제대로 인식하지 못하여 변수명 재설정.()
df.columns = list(df.columns)[:-3] + [''을금실_개수', '시군구별_인구', 'Risk']
df1 = df.drop(['사망자수', '중상자수', '경상자수', '부상신고자수'], axis = 1)
```

```
In [9]: linear_model = smf.ols(formula = 'Risk ~ ' + '+'.join(df1.columns[:-1]), data = df1)
result = linear_model.fit()
result.summary()
```

OLS Regression Results								
Dep. Variable:	Risk		R-squared:	0.217				
Model:	OLS		Adj. R-squared:	0.216				
Method:	Least Squares			F-statistic:	132.8			
Date:	Tue, 17 Aug 2021		Prob (F-statistic):	0.00				
Time:	08:12:38		Log-Likelihood:	-36973.				
No. Observations:	12448			AIC:	7.400e+04			
Df Residuals:	12421			BIC:	7.420e+04			
Df Model:	26							
Covariance Type:	nonrobust							
	coef	std err	t	P> t	[0.025]			
Intercept	-1.0692	0.500	-2.139	0.032	-2.04			
발생시간	0.0160	0.007	2.262	0.024	0.00			
발생지 시도	0.0458	0.009	5.367	0.000	0.00			

발생지_시군구	0.0027	0.001	2.816	0.005	0.001	0.005
요일	0.0720	0.021	3.442	0.001	0.031	0.113
사고내용	1.9210	0.044	43.894	0.000	1.835	2.007
사고유형_대분류	-1.2494	0.296	-4.221	0.000	-1.830	-0.669
사고유형_중분류	0.0194	0.009	2.133	0.033	0.002	0.037
가해자법규위반	0.0385	0.046	0.835	0.404	-0.052	0.129
가해자성별	0.2356	0.116	2.035	0.042	0.009	0.462
가해자연령	0.0313	0.028	1.138	0.255	-0.023	0.085
가해당사자종별	0.1067	0.052	2.052	0.040	0.005	0.209
가해자신체상해정도	-0.4889	0.036	-13.426	0.000	-0.560	-0.417
피해자성별	-0.2308	0.056	-4.139	0.000	-0.340	-0.122
피해자연령	0.1038	0.031	3.393	0.001	0.044	0.164
피해당사자종별	-0.1355	0.049	-2.758	0.006	-0.232	-0.039
피해자신체상해정도	-0.2313	0.024	-9.468	0.000	-0.279	-0.183
가해차량용도_대분류	0.7792	0.096	8.102	0.000	0.591	0.968
가해차량용도_중분류	-0.1783	0.027	-6.576	0.000	-0.231	-0.125
가해차량용도	0.1187	0.027	4.378	0.000	0.066	0.172
피해차량용도_대분류	0.9703	0.123	7.894	0.000	0.729	1.211
피해차량용도_중분류	-0.1220	0.026	-4.688	0.000	-0.173	-0.071
피해차량용도	0.1283	0.024	5.431	0.000	0.082	0.175
기상상태	0.1001	0.079	1.274	0.203	-0.054	0.254
노면상태	0.0621	0.041	1.518	0.129	-0.018	0.142
응급실_개수	-0.0235	0.016	-1.434	0.152	-0.056	0.009
시군구별_인구	-3.221e-07	1.58e-07	-2.033	0.042	-6.33e-07	-1.15e-08

Omnibus: 18394.266 Durbin-Watson: 1.948

Prob(Omnibus): 0.000 Jarque-Bera (JB): 15206092.117

Skew: 8.730 Prob(JB): 0.00

Kurtosis: 173.331 Cond. No. 6.52e+06

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 6.52e+06. This might indicate that there are strong multicollinearity or other numerical problems.

## 다중공선성 확인

```
In [10]: df2 = df1.drop(['Risk'], axis=1)
vif1 = pd.DataFrame()
vif1["Features"] = df2.columns
vif1["VIF"] = [variance_inflation_factor(df2.values, i) for i in range(df2.shape[1])]
vif1.sort_values("VIF", inplace=True, ascending=False)
vif1.round(3)
```

	Features	VIF
19	피해차량용도_대분류	84.472
21	피해차량용도	57.636
18	가해차량용도	55.407
5	사고유형_대분류	53.982
20	피해차량용도_중분류	43.907
14	피해당사자종별	43.529
17	가해차량용도_중분류	39.430
10	가해당사자종별	34.362
16	가해차량용도_대분류	25.187
7	가해자법규위반	21.384
22	기상상태	15.979

11	가해자신체상해정도	10.071
8	가해자성별	9.416
13	피해자연령	6.549
0	발생시간	5.292
9	가해자연령	4.976
2	발생지_시군구	4.710
12	피해자성별	4.489
25	시군구별_인구	4.132
6	사고유형_중분류	4.032
15	피해자신체상해정도	3.193
3	요일	3.149
4	사고내용	3.108
24	응급실_개수	2.821
1	발생지_시도	1.992
23	노면상태	1.542

## 후진제거법(Backward Elimination)

- 모든 변수가 포함된 모형으로부터 불필요한 독립변수들을 하나씩 제거해 나가는 과정을 반복하며 모형을 단순화해 나가는 방법.
- 설명변수를 모두 포함한 완전모형(Full model)에서 설명력(상관)이 가장 작은 설명변수부터 하나씩 제거하는 방법.

In [11]:

```
Target = df1['Risk']
Feature = df1.iloc[:, :-1]
print(Feature.shape, Target.shape)
```

(12448, 26) (12448,)

In [12]:

```
model = LinearRegression()
rfe = RFE(estimator = model, n_features_to_select = int(Feature.shape[-1] / 2)).fit(Feature, Target)
selected_cols = Feature.columns[rfe.support_]
removed_cols = Feature.columns[~rfe.support_]
print(f'Selected Variables : {list(selected_cols)}')
print()
print(f'Removed Variables : {list(removed_cols)}')
```

Selected Variables : ['사고내용', '사고유형\_대분류', '가해자성별', '가해자신체상해정도', '피해자성별', '피해당사자종별', '피해자신체상해정도', '가해차량용도\_대분류', '가해차량용도\_중분류', '가해차량용도', '피해차량용도\_대분류', '피해차량용도', '기상상태']

Removed Variables : ['발생시간', '발생지\_시도', '발생지\_시군구', '요일', '사고유형\_중분류', '가해자범규위반', '가해자연령', '가해당사자종별', '피해자연령', '피해차량용도\_중분류', '노면상태', '응급실\_개수', '시군구별\_인구']

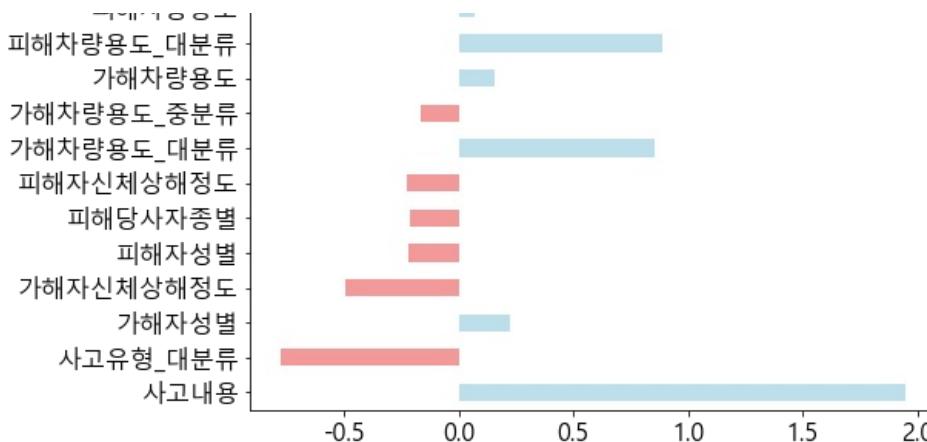
## 종속 변수에 영향을 주는 중요 변수 파악

- 양의 영향을 주는 변수
  - [사고내용, 피해차량용도\_대분류, 가해차량용도\_대분류, 가해자성별, 가해차량용도, 피해자차량용도]
- 음의 영향을 주는 변수
  - [사고유형\_대분류, 가해자 신체상해정도, 피해자성별, 피해자신체상해정도, 피해당사자종별, 가해차량용도\_중분류]

In [13]:

```
rfe_linear_model = smf.ols(formula = 'Risk~' + '+'.join(list(selected_cols[:-1])), data = df1)
rfe_linear_model.fit().summary()

linear_coef = pd.DataFrame({'Cofficient' : rfe_linear_model.fit().params.values[1:]},
                           index = selected_cols[:-1])
linear_coef['Positive'] = linear_coef['Cofficient'] > 0
linear_coef['Cofficient'].plot.barh(y = "Cofficient", legend = False, alpha = .8,
                                     color = linear_coef['Positive'].map({True : 'lightblue', False : 'lightcoral'}))
plt.show()
```



### 표준화 회귀계수 사용

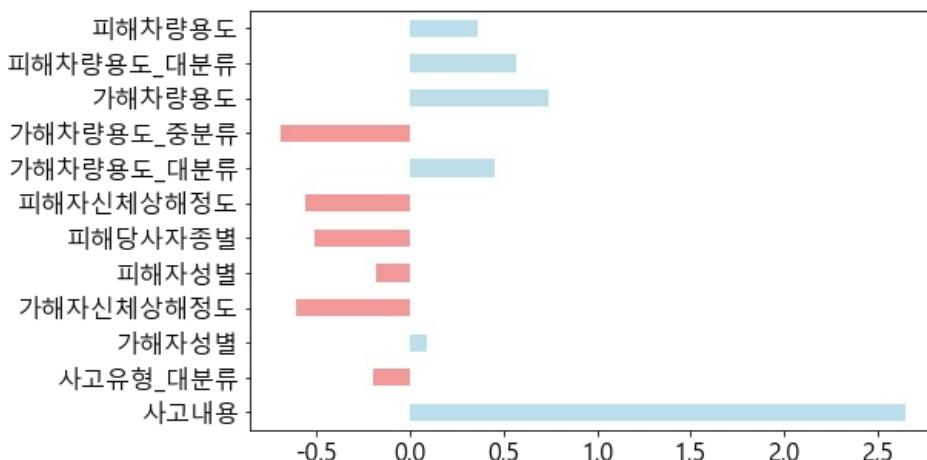
- 각 변수에 대한 단위가 서로 다르므로 이에 대한 올바른 영향력을 평가하기 위해 데이터를 정규화\*(Scaling)하여 이용.

In [14]:

```
scaler = StandardScaler()
cols = Feature.columns
np_scaled = scaler.fit_transform(Feature)
df_scaled = pd.DataFrame(np_scaled, columns=cols)
df_scaled['Risk'] = Target
# df_scaled.head()

rfe_linear_scaled_model = smf.ols(formula = 'Risk~' + '+'.join(list(selected_cols[:-1])), data = df_scaled)
rfe_linear_scaled_model.fit().summary()

linear_scaled_coef = pd.DataFrame({"Cofficient" : rfe_linear_scaled_model.fit().params.values[1:],
                                    index = selected_cols[:-1])
linear_scaled_coef['Positive'] = linear_coef['Cofficient'] > 0
linear_scaled_coef.plot.barh(y = "Cofficient", legend = False, alpha = .8,
                             color = linear_scaled_coef['Positive'].map({True : 'lightblue', False : 'lightcoral'})
plt.show()
```



### 다중공선성 재확인

- 위 lambda 계수를 통해 살펴본 다중공선성을 일으킬 가능성이 있는 변수 목록안에 VIF 값이 15 이상인 것이 존재.
  - [발생시간, 발생시\_시군구, 사고유형\_대분류, 가해자법규위반, 가해자성별, 가해 당사자 종별, 피해자 성별, 피해당사자종별, 가해자차량용도\_중분류, 피해자차량용도\_대분류, 피해차량용도\_대분류, 기상상태, 가해자차량용도, 피해차량용도, 피해자차량용도\_중분류]

In [15]:

```
df3 = df1[list(selected_cols)]
vif2 = pd.DataFrame()
vif2["Features"] = df3.columns
vif2["VIF"] = [variance_inflation_factor(df3.values, i) for i in range(df3.shape[1])]
vif2.sort_values("VIF", inplace=True, ascending = False)
vif2.round(3)
```

Out[15]:

	Features	VIF
10	피해차량용도_대분류	75.896
1	사고유형_대분류	48.625

11	피해차량용도	37.187
5	피해당사자종별	36.888
9	가해차량용도	36.881
8	가해차량용도_중분류	35.233
7	가해차량용도_대분류	21.477
12	기상상태	11.233
3	가해자신체상해정도	9.709
2	가해자성별	8.674
4	피해자성별	4.341
6	피해자신체상해정도	3.129
0	사고내용	3.086

- 가해차량용도\_대분류, 피해차량용도\_대분류 변수는 다중공선성의 문제를 일으켜 추후 학습시 과적합을 야기할 가능성도 있지만, 동시에 종속변수(Target)에 큰 영향을 끼치는 변수이므로 포함시켜보기로 함.

```
In [83]: df4 = df1[['기상상태', '가해자신체상해정도', '가해자성별', '피해자성별', '피해자신체상해정도', '사고내용', '노면상태', '발생지_시도', '발생시간', '피해차량용도_중분류', '가해차량용도', '응급실_개수', '시군구별_인구', '피해자연령', '요일']]
df5 = df[list(df4.columns) + ['사망자수', '중상자수', '경상자수', '부상신고자수']]

vif3 = pd.DataFrame()
vif3["Features"] = df5.columns
vif3["VIF"] = [variance_inflation_factor(df5.values, i) for i in range(df5.shape[1])]
vif3.sort_values("VIF", inplace=True, ascending=False)
vif3.round(3)
```

	Features	VIF
9	피해차량용도_대분류	27.721
8	가해차량용도_대분류	21.198
18	가해자법규위반	21.190
0	기상상태	15.855
1	가해자신체상해정도	10.445
2	가해자성별	9.300
10	가해차량용도	7.025
17	피해차량용도_중분류	6.604
13	피해자연령	6.192
16	발생시간	5.252
15	가해자연령	4.863
7	발생지_시군구	4.697
5	사고내용	4.449
3	피해자성별	4.285
12	시군구별_인구	4.153
19	사고유형_중분류	4.031
4	피해자신체상해정도	3.379
14	요일	3.152
11	응급실_개수	2.816
22	중상자수	2.542
23	경상자수	2.085
20	발생지_시도	2.004
6	노면상태	1.533
21	사망자수	1.121
24	부상신고자수	1.089

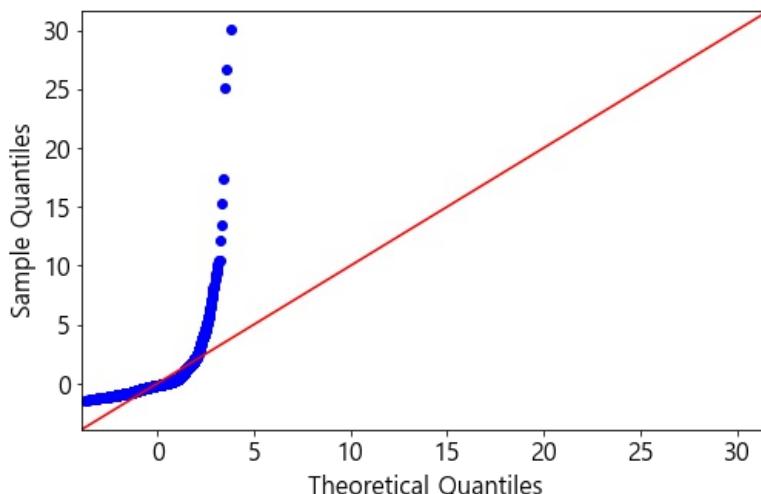
## 잔차 확인

- Q-Q plot을 통한 잔차의 정규성을 확인.
- 잔차의 형태가 비선형을 보이는 것으로 보아 정규성을 만족하고 있다고 판단하기 어렵다.

- 따라서 위 모델은 다중 회귀 분석의 가정을 위배하고 있으므로, 적절한 판단의 근거로 사용하기 어렵다고 볼 수 있다.
- 오차의 정규성과 등분산성을 고려치 않아도 되는 일반화 선형 회귀(Generalized Linear Regression)를 이용토록 한다.

In [17]:

```
fig = sm.qqplot(result.resid, fit = True, line = '45')
```



## 포아송 회귀 모형

- 반응변수 Risk를 사고 건수와 같은 Count 지표로서 고려한 포아송 회귀 분석을 이용.
- 포아송 분포는 분산과 평균이 모두 같다는 성질을 가짐.
- 진단문제는 포아송 분포로 예측되는 것보다 종속변수의 분산이 클 때 발생할 수 있음.
- 따라서 과대산포 검정 시행한 결과 p-value가 유의수준 5%하에서 과대산포가 아닐 것이라는 귀무가설을 기각하므로 과대산포가 발생했다고 볼 수 있음.
- 이와 같이 과대산포(Overdispersion) 문제가 있는 경우, 이를 해결하지 않으면 모형에서 유의성 검사 결과를 믿을 수 없으므로 유사포아송(Quasipoisson) 이용 혹은 다른 대안을 찾는 것이 적절하다고 판단함.

In [5]:

```
glm_poisson = glm(Risk ~ ., df, family = poisson(link="log"))
summary(glm_poisson)
```

Call:

```
glm(formula = Risk ~ ., family = poisson(link = "log"), data = df)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-26.0517	-0.8023	-0.5128	0.3045	7.0402

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	-1.241e+02	3.878e+01	-3.201	0.00137 **
X	-1.774e-05	4.373e-06	-4.057	4.97e-05 ***
발생일	6.169e-06	1.923e-06	3.208	0.00134 **
발생시간	4.476e-03	7.590e-04	5.897	3.71e-09 ***
발생지_시도	8.571e-03	8.811e-04	9.728	< 2e-16 ***
발생지_시군구	8.236e-04	1.042e-04	7.903	2.71e-15 ***
요일	9.489e-03	2.285e-03	4.153	3.29e-05 ***
사고내용	3.716e-01	4.365e-03	85.120	< 2e-16 ***
사고유형_대분류	-3.271e-02	2.991e-02	-1.093	0.27423
사고유형_중분류	1.123e-02	1.037e-03	10.824	< 2e-16 ***
가해자별규위반	-3.288e-03	4.858e-03	-0.677	0.49852
가해자성별	8.943e-02	1.300e-02	6.881	5.96e-12 ***
가해자연령	1.389e-02	3.007e-03	4.620	3.85e-06 ***
가해당사자종별	6.903e-02	5.123e-03	13.475	< 2e-16 ***
가해자신체상해정도	-6.872e-02	3.526e-03	-19.493	< 2e-16 ***
피해자성별	-2.111e-02	6.472e-03	-3.263	0.00110 **
피해자연령	1.093e-02	3.318e-03	3.293	0.00099 ***
피해당사자종별	1.882e-02	4.819e-03	3.906	9.38e-05 ***
피해자신체상해정도	-2.224e-02	2.262e-03	-9.834	< 2e-16 ***
가해차량용도_대분류	1.878e-02	1.024e-02	1.833	0.06674 .
가해차량용도_중분류	-1.521e-02	2.797e-03	-5.439	5.36e-08 ***
가해차량용도	-1.247e-02	2.682e-03	-4.650	3.31e-06 ***
피해차량용도_대분류	6.764e-02	1.255e-02	5.391	7.01e-08 ***
피해차량용도_중분류	-1.950e-02	2.871e-03	-6.791	1.11e-11 ***
피해차량용도	4.341e-03	2.505e-03	1.733	0.08313 .
기상상태	3.022e-05	8.019e-03	0.004	0.99699
노면상태	2.740e-03	4.250e-03	0.645	0.51910
사망자수	7.016e-01	6.098e-03	115.057	< 2e-16 ***

```

중상자수      6.732e-02  1.777e-03  37.885 < 2e-16 ***
경상자수      3.858e-02  9.310e-04  41.437 < 2e-16 ***
부상신고자수  3.906e-02  1.713e-03  22.801 < 2e-16 ***
응급실_개수   2.972e-03  1.846e-03  1.610  0.10742
시군구별_인구 -8.687e-08  1.791e-08 -4.850  1.23e-06 ***
...
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

(Dispersion parameter for poisson family taken to be 1)

```

Null deviance: 48593  on 12447  degrees of freedom
Residual deviance: 14150  on 12415  degrees of freedom
AIC: 49543

```

Number of Fisher Scoring iterations: 6

In [ ]:

```
glm_poisson = glm(Risk ~ ., df, family = poisson(link="log"))
summary(glm_poisson)
```

## 과대산포 과정

- **object** : a fitted Poisson GLM of class "glm" as fitted by glm with family poisson.
- **trafo** : a specification of the alternative (see also details), can be numeric or a (positive) function or NULL (the default).
- **alternative** : a character string specifying the alternative hypothesis: "greater" corresponds to overdispersion, "less" to underdispersion and "two.sided" to either one.

$$H_0: c = 1 \quad H_1: c > 1$$

In [16]:

```
dispersiontest(glm_poisson)
```

Overdispersion test

```

data: glm_poisson
z = 6.4609, p-value = 5.205e-11
alternative hypothesis: true dispersion is greater than 1
sample estimates:
dispersion
1.323801

```

## 모델링(Modeling)

- 2017 ~ 2018년 데이터 = Train dataset
- 2019년 데이터 = Test dataset : 검증용이자, 미래 Risk(위험도) 예측용

## 데이터셋 분할하기

- Train : Test = 66 : 34

In [104]:

```
df_scaled
```

Out[104]:

	발생시간	발생지_시도	발생지_시군구	요일	사고내용	사고유형_대분류	사고유형_중분류	가해자법 규위반	가해자성별	가해자연령	...	가해차량 용도_중분류	가해차량 용도	피해 용도
0	-2.078625	-0.338750	-1.346069	0.487983	1.449666	-0.082658	0.585652	0.510556	-0.214442	-0.711803	...	-0.459190	-0.399622	-0.404
1	-1.751420	1.781873	1.555550	0.487983	0.713928	-0.082658	0.585652	0.510556	-0.214442	-0.111578	...	-0.459190	-0.399622	-0.404
2	-1.587817	-0.338750	-0.341663	0.487983	-0.757547	-0.082658	-1.500539	0.510556	-0.214442	-0.711803	...	0.023417	0.446381	1.158
3	-1.424215	0.625170	-1.212148	0.487983	-0.021810	-0.082658	0.794271	-2.730894	-2.776222	2.289323	...	-0.700493	-0.611123	1.158
4	-0.933407	-0.917102	-1.837112	0.487983	-0.757547	-0.082658	0.585652	0.510556	2.347338	-1.312028	...	-0.459190	-0.399622	-0.404
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12443	0.866221	0.046818	-0.921986	1.471536	1.449666	-0.082658	0.794271	0.510556	-0.214442	-0.111578	...	-0.459190	-0.399622	-0.404
12444	1.029823	-0.724318	1.399309	1.471536	-0.757547	-0.082658	-1.500539	0.510556	-0.214442	1.689098	...	-0.459190	-0.399622	1.158
12445	1.029823	-0.724318	1.399309	1.471536	-0.757547	-0.082658	-1.500539	0.510556	-0.214442	1.689098	...	-0.459190	-0.399622	-0.404
12446	1.029823	-0.724318	1.600190	1.471536	-0.757547	-0.082658	-1.500539	-0.569927	-0.214442	-1.312028	...	-0.459190	-0.399622	1.158
12447	1.193426	-0.724318	1.600190	1.471536	-0.757547	-0.082658	-1.500539	-0.569927	-0.214442	0.488647	...	-0.459190	-0.399622	1.158

12448 rows × 27 columns

```
In [121]:
```

```
x_data = df_scaled[list(df4.columns)]
y_data = df_scaled.iloc[:, -1]
# x_train, x_test, y_train, y_test = train_test_split(df_scaled, y_data, test_size = 0.3, random_state = 2021)
# 2017 ~ 2018
x_train = df[:8225].iloc[:, :-1]
y_train = df[:8225].iloc[:, -1]

# 2019
x_test = df[8225:8225].iloc[:, :-1]
y_test = df[8225:8225].iloc[:, -1]
print(f'train data x size : {x_train.shape}')
print(f'train data y size : {y_train.shape}')
print(f'test data x size : {x_test.shape}')
print(f'test data y size : {y_test.shape}')
print()
print(f'test data ratio : {round(len(x_test) / len(df), 2) * 100}%')
```

```
train data x size : (8225, 30)
train data y size : (8225,)
test data x size : (4223, 30)
test data y size : (4223,)

test data ratio : 34.0%
```

```
In [ ]:
```

```
x_data = df_scaled[list(df4.columns)]
y_data = df_scaled.iloc[:, -1]
# x_train, x_test, y_train, y_test = train_test_split(df_scaled, y_data, test_size = 0.3, random_state = 2021)
# 2017 ~ 2018
x_train = x_data[:8225].iloc[:, :-1]
y_train = y_data[:8225].iloc[:, -1]

# 2019
x_test = x_data[8225:8225].iloc[:, :-1]
y_test = y_data[8225:8225].iloc[:, -1]
print(f'train data x size : {x_train.shape}')
print(f'train data y size : {y_train.shape}')
print(f'test data x size : {x_test.shape}')
print(f'test data y size : {y_test.shape}')
print()
print(f'test data ratio : {round(len(x_test) / len(df), 2) * 100}%')
```

## GridSearchCV 함수 생성

- 관심 있는 매개변수들을 대상으로 가능한 모든 조합을 시도하여 최적의 매개변수를 찾는 방법.

```
In [122]:
```

```
Regressors = []
Parameters = []

def gridSearchCV(models, params, count) :
    best_models = []
    for i in tqdm(range(0, count)):
        model_grid = GridSearchCV(models[i], Parameters[i], cv = KFold(n_splits = 5), scoring = 'r2' )
        model_grid.fit( x_train, y_train )
        best_models.append( model_grid.best_estimator_ )
    return best_models
```

## Hyperparameter tuning

### DecisionTreeRegressor

- 일반적으로 의사 결정 나무(decision tree)는 여러 가지 규칙을 순차적으로 적용하면서 독립 변수 공간을 분할하는 분류 모형.
- 분류(classification)와 회귀 분석(regression)에 모두 사용될 수 있기 때문에 CART(Classification And Regression Tree)라고도 함.
- 전체 학습 데이터 집합(부모 노드)을 해당 독립 변수의 값이 기준값보다 작은 데이터 그룹(자식 노드 1)과 해당 독립 변수의 값이 기준값보다 큰 데이터 그룹(자식 노드 2)으로 나눔.
- 각각의 자식 노드에 대해 1~2의 단계를 반복하여 하위의 자식 노드를 만든다. 단, 자식 노드에 한가지 클래스의 데이터만 존재한다면 더 이상 자식 노드를 나누지 않고 중지.
- 자식 노드 나누기를 연속적으로 적용하면 노드가 계속 증가하는 나무(tree)와 같은 형태.

```
In [123]:
```

```
DTreg      = DecisionTreeRegressor(random_state = 2021)
Parameters_DR = {"criterion" : ["friedman_mse", "mae"],
                 "max_features" : ["auto", "sqrt", "log2"],
                 "max_depth" : list(range(3, 5)),
```

```
        "min_samples_split" : list(range(4, 20, 3)),
        "min_samples_leaf"  : list(range(3, 5))
    }
Regressors.append(DTreg)
Parameters.append(Parameters_DB)
```

## RandomForestRegressor

- Classification(분류) 및 Regression(회귀) 문제에 모두 사용 가능.
  - Missing value(결측치)를 다루기 쉬움.
  - 대용량 데이터 처리에 효과적.
  - 모델의 노이즈를 심화시키는 Overfitting(오버피팅) 문제를 회피하여, 모델 정확도를 향상시킴.
  - Classification 모델에서 상대적으로 중요한 변수를 선정 및 Ranking 가능.

```
In [124]: RFreg      = RandomForestRegressor(oob_score=True, random_state = 2021)
Parameters_RF = {"max_features" : list(range(4, 11)),
                 "n_estimators"  : list(range(100, 500, 100))
                }
Regressors.append(RFreg)
Parameters.append(Parameters_RF)
```

## GradientBoostingRegressor

- 여러 개의 결정 트리를 묶어 강력한 모델을 만드는 또 다른 앙상블 기법.
  - 회귀와 분류에 모두 사용할 수 있음.
  - 랜덤포레스트와 달리 이진 트리의 오차를 보완하는 방식으로 순차적으로 트리를 만듬.
  - 무작위성이 없고 강력한 사전 가지치기가 사용됨.
  - 1~5개의 깊지 않은 트리를 사용하기 때문에 메모리를 적게 사용하고 예측이 빠름.
  - **learning\_rate** : 오차를 얼마나 강하게 보정할 것인지를 제어.
  - **n\_estimator**의 값을 키우면 앙상블에 트리가 더 많이 추가되어 모델의 복잡도가 커지고 훈련 세트에서의 실수를 바로잡을 기회가 많아지지-만, 너무 크면 모델이 복잡해지고 오버피팅(과대적합)이 될 수 있음.
  - **maxdepth(maxleaf nodes)** 복잡도를 너무 높이지 말고 트리의 깊이가 5보다 깊어지지 않게 함.

```
In [125]: GBreg = GradientBoostingRegressor(random_state = 2021, min_samples_split = 24)
Parameters_GB = {"n_estimators" : list(range(60, 75)),
                 "min_samples_leaf" : list(range(60, 100, 20)),
                 "max_depth" : list(range(3, 5)),
                 "learning_rate" : [10.0 ** i for i in range(-3, -0)]}
}
Regressors.append(GBreg)
Parameters.append(Parameters_GB)
```

```
In [126]: best_model_list = gridSearchCV( Regressors, Parameters, len(Regressors) )
```

## Voting

- 여러 종류의 알고리즘을 사용한 각각의 결과에 대해 투표를 통해 최종 결과를 예측하는 방식

```
In [127]: Vtreg = VotingRegressor( estimators = [ ('DT', best_model_list[1]),  
                                             ('RF', best_model_list[2]),  
                                             ('GB', best_model_list[-1])]  
Vtreg.fit(x_train, y_train)
```

# 모델 평가

평가지표	설명
R-Squared	분산을 기반으로 예측 성능을 평가하는 지표이며, 추가되는 변수가 많을수록 값이 계속 커져 모델 성능을 왜곡할 수 있다.
Adjusted R-squared	결정계수의 단점을 보완한 지표로서 실제로 종속변수에 영향을 주는 독립변수들에 의해 설명되는 분산의 비율을 통해 평가된다.
MAE(Mean Absolute Error)	매우 직관적인 지표이나, 스케일에 의존적이다.
MSE(Mean Squared Error)	지표 자체가 직관적이고 단순하다는 장점이 있으나, 스케일에 의존적이며 값을 왜곡시킬 수 있다.
RMSE(Root Mean Squared Error)	MSE의 성질을 기본적으로 따르지만, 루트를 통해 제곱값을 풀어주므로 상대적으로 왜곡의 정도가 덜하다.
MAPE (Mean Absolute Percentage Error)	평균 절대 백분율 오차(비율 에러)라고 하며 백분율로 표시하고 크기 의존적 에러의 단점을 커버하기 위한 지표이다.

## Risk 예측모델 평가함수 생성

```
In [128...]  
R2 = lambda y_true, y_pred : metrics.r2_score(y_true, y_pred)  
AdjR2 = lambda y_true, y_pred, x_true : 1 - (1 - metrics.r2_score(y_true, y_pred)) * (len(y_true)-1) / (len(y_true)-2)  
MAE = lambda y_true, y_pred : metrics.mean_absolute_error(y_true, y_pred)  
MSE = lambda y_true, y_pred : metrics.mean_squared_error(y_true, y_pred)  
RMSE = lambda y_true, y_pred : np.sqrt(metrics.mean_squared_error(y_true, y_pred))  
MAPE = lambda y_true, y_pred : np.mean(np.abs((y_true - y_pred) / y_true)) * 100  
  
def Evaluate(model, y_test, x_test) :  
    r2 = R2(y_test, model.predict(x_test))  
    adjr2 = AdjR2(y_test, model.predict(x_test), x_test)  
    mae = MAE(y_test, model.predict(x_test))  
    mse = MSE(y_test, model.predict(x_test))  
    rmse = RMSE(y_test, model.predict(x_test))  
    mape = MAPE(y_test, model.predict(x_test))  
  
    # print(f'best estimator model R-squared : {r2}')  
    # print(f'best estimator model Adj-R-squared : {adjr2}')  
    # print(f'best estimator model MAE : {mae}')  
    # print(f'best estimator model MSE : {mse}')  
    # print(f'best estimator model RMSE : {rmse}')  
    # print(f'best estimator model MAPE : {mape}')  
    return r2, adjr2, mae, mse, rmse, mape  
  
models_name = ['DT', 'RF', 'GB', 'VT']
```

## Train 데이터 평가

```
In [129...]  
model_list = best_model_list + [Vtreg]  
evaluate_df = pd.DataFrame()  
for idx, name in enumerate(models_name) :  
    evaluate_df[f'{name}'] = Evaluate(model_list[idx], y_train, x_train)  
evaluate_df.index = ['R-squared', 'Adj-R-squared', 'MAE', 'MSE', 'RMSE', 'MAPE']  
evaluate_df
```

	DT	RF	GB	VT
R-squared	0.845862	0.984130	0.842327	0.909568
Adj-R-squared	0.845298	0.984072	0.841750	0.909237
MAE	1.055551	0.081107	0.411154	0.287378
MSE	4.724257	0.486405	4.832610	2.771701
RMSE	2.173536	0.697428	2.198320	1.664843
MAPE	36.747718	1.086018	7.942533	5.490381

## Test 데이터 평가

```
In [130...]  
model_list = best_model_list + [Vtreg]  
evaluate_df = pd.DataFrame()  
for idx, name in enumerate(models_name) :  
    evaluate_df[f'{name}'] = Evaluate(model_list[idx], y_test, x_test)  
evaluate_df.index = ['R-squared', 'Adj-R-squared', 'MAE', 'MSE', 'RMSE', 'MAPE']  
evaluate_df
```

	DT	RF	GB	VT
R-squared	0.817864	0.927052	0.851601	0.891001

<b>Adj-R-squared</b>	0.816560	0.926530	0.850539	0.890221
<b>MAE</b>	1.045843	0.181587	0.362518	0.268909
<b>MSE</b>	4.381495	1.754843	3.569911	2.622106
<b>RMSE</b>	2.093202	1.324705	1.889421	1.619292
<b>MAPE</b>	37.975556	2.787248	8.085739	5.909932

In [131...]

```
f, ax = plt.subplots(3, 2, figsize = (10, 10))
plt.tight_layout(w_pad = 5, h_pad = 5)

ax[0,0].bar(models_name, list(evaluate_df.loc['R-squared'].values))
ax[0,0].set_title("R-squared").set_fontsize(15)
ax[0,0].set_xticklabels(models_name, rotation=90)

ax[0,1].bar(models_name, list(evaluate_df.loc['Adj-R-squared'].values))
ax[0,1].set_title("Adj-R-squared").set_fontsize(15)
ax[0,1].set_xticklabels(models_name, rotation=90)

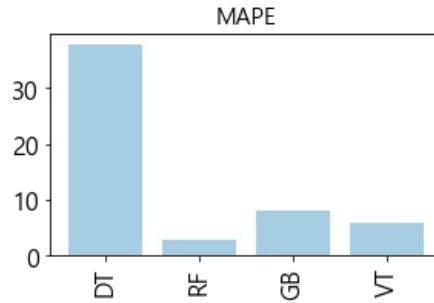
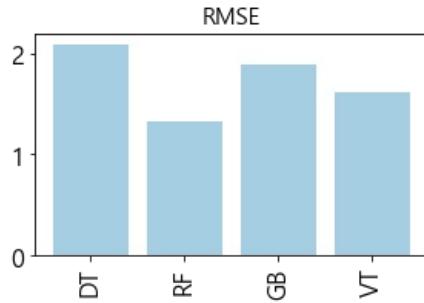
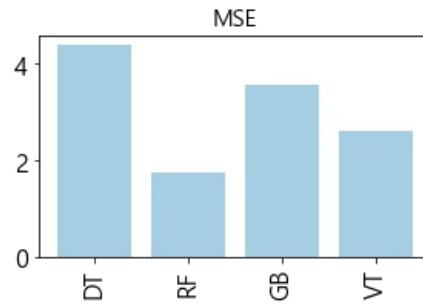
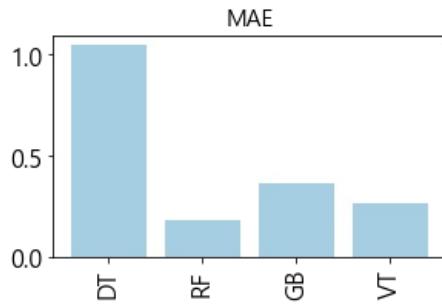
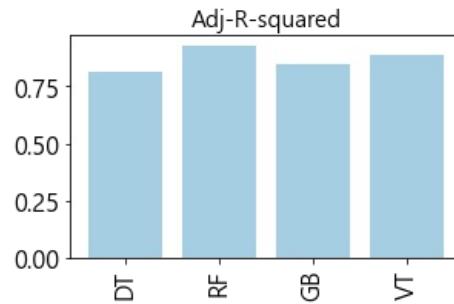
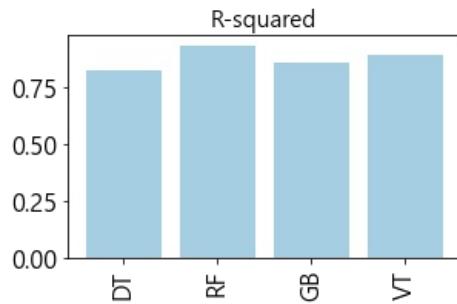
ax[1,0].bar(models_name, list(evaluate_df.loc['MAE'].values))
ax[1,0].set_title("MAE").set_fontsize(15)
ax[1,0].set_xticklabels(models_name, rotation=90)

ax[1,1].bar(models_name, list(evaluate_df.loc['MSE'].values))
ax[1,1].set_title("MSE").set_fontsize(15)
ax[1,1].set_xticklabels(models_name, rotation=90)

ax[2,0].bar(models_name, list(evaluate_df.loc['RMSE'].values))
ax[2,0].set_title("RMSE").set_fontsize(15)
ax[2,0].set_xticklabels(models_name, rotation=90)

ax[2,1].bar(models_name, list(evaluate_df.loc['MAPE'].values))
ax[2,1].set_title("MAPE").set_fontsize(15)
ax[2,1].set_xticklabels(models_name, rotation=90)
```

Out[131] [Text(0, 0, 'DT'), Text(1, 0, 'RF'), Text(2, 0, 'GB'), Text(3, 0, 'VT')]



## 최종모델의 학습시 중요 변수

- 상위 10개 중요 변수 선정.
- 최종 모델 RandomForest 모델 사용.

In [136...]

```
feature_name = x_train.columns
df_importance = pd.DataFrame()
df_importance["Feature"] = feature_name
df_importance["Importance"] = best_model_list[1].feature_importances_
df_importance.sort_values("Importance", ascending=False, inplace=True)
df_importance.round(3)[:10]
```

Out[136...]

	Feature	Importance
25	중상자수	0.271
24	사망자수	0.233
26	경상자수	0.190
4	사고내용	0.136
27	부상신고자수	0.060
15	피해자신체상해정도	0.020
11	가해자신체상해정도	0.009
2	발생지_시군구	0.008
29	시군구별_인구	0.007
17	가해차량용도_중분류	0.006

## 라벨링 변수 다시 전환

In [138...]

```
df_re = pd.read_csv('total_data.csv')
df_re.head()
```

Out[138...]

Unnamed: 0	발생일	발생시간	발생지_시도	발생지_시군구	요일	사고내용	사고유형_대분류	사고유형_중분류	가해자법규위반	...	피해차량용도	기상상태	노면상태	사망자수	중상자수	경상자수	부상신고자수	응급실개수	시군구별_인구	Risk
0	0	20170101 00시	경북	구미시	일	중상	차대차	추돌	안전운전 의무불이행	...	승용차	맑음	건조	0	8	0	1	3	421012.0	25
1	1	20170101 02시	충남	홍성군	일	사망	차대차	추돌	안전운전 의무불이행	...	승용차	맑음	건조	1	0	1	0	1	101025.0	13
2	2	20170101 03시	경북	상주시	일	경상	차대차	기타	안전운전 의무불이행	...	렌터카	맑음	건조	0	0	2	0	2	100644.0	2
3	3	20170101 04시	서울	금천구	일	부상신고	차대차	측면충돌	기타	...	렌터카	맑음	젖음/습기	0	0	0	1	1	233960.0	1
4	4	20170101 07시	강원	강릉시	일	경상	차대차	추돌	안전운전 의무불이행	...	승용차	맑음	건조	0	0	4	0	4	213450.0	4

5 rows × 33 columns

## 실제 데이터

In [143...]

```
df_index_train = list(df_re['발생지_시군구'][:8225].values)
df_train_result = pd.DataFrame()
df_train_result['Index'] = df_index_train
df_train_result['Risk'] = y_train
df_train_result['응급실_개수'] = df_re['응급실_개수'][:8225]
df_train_result['시군구별_인구'] = df_re['시군구별_인구'][:8225].astype('int64')
df_train_result.columns = ['발생지_시군구', 'Risk', '응급실_개수', '시군구별_인구']
df_train_result.to_csv('train1.csv', index=False)
df_train_result
```

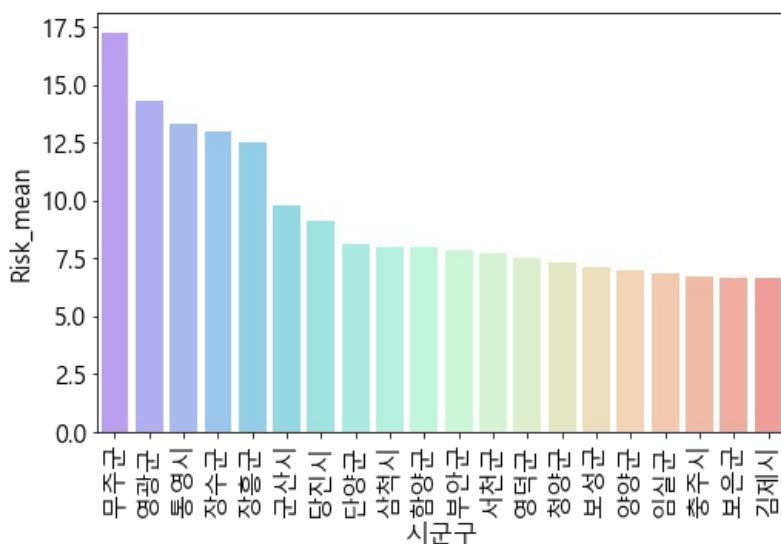
Out[143...]

	발생지_시군구	Risk	응급실_개수	시군구별_인구
0	구미시	25	3	421012
1	홍성군	13	1	101025

2	상주시	2	2	100644
3	금천구	1	1	233960
4	강릉시	4	4	213450
...	...	...	...	...
8220	군위군	6	0	23992
8221	유성구	3	1	349197
8222	용인시	6	4	1032939
8223	군위군	3	0	23992
8224	천안시	2	4	643288

8225 rows × 4 columns

```
In [140...]: top20 = df_train_result.groupby('발생지_시군구').mean().sort_values(by='Risk', ascending=False)[:20]
sb.barplot(top20.index, [i[0] for i in top20.values], palette='rainbow', alpha=.5)
plt.xticks(rotation = 90)
plt.xlabel('시군구')
plt.ylabel('Risk_mean')
plt.show()
```



## 예측 데이터

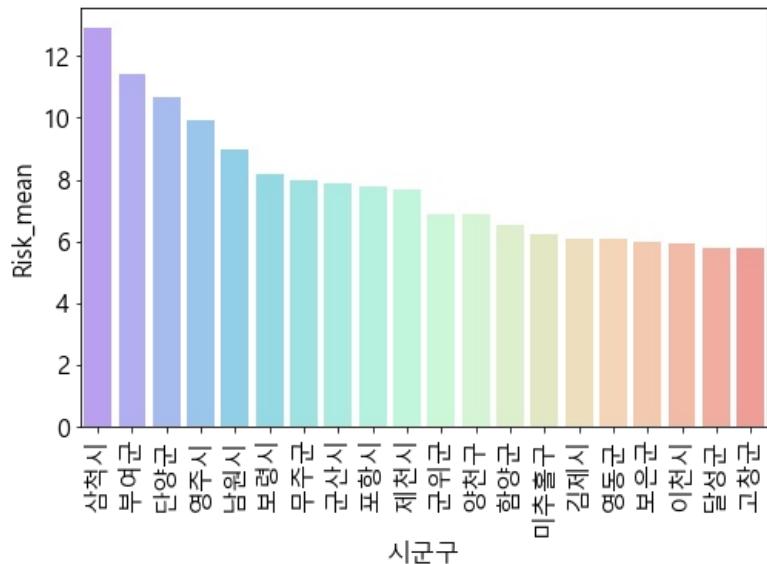
```
In [144...]: df_index = list(df_re['발생지_시군구'][8225:].values)
y_pred = Vtreg.predict(x_test)
df_test_result = pd.DataFrame()
df_test_result['Index'] = df_index
df_test_result['Risk'] = y_pred
df_test_result['응급실_개수'] = list(df_re['응급실_개수'][8225:].values)
df_test_result['시군구별_인구'] = list(df_re['시군구별_인구'][8225:].astype('int64').values)
df_test_result.columns = ['발생지_시군구', 'Risk', '응급실_개수', '시군구별_인구']
df_test_result.to_csv('test1.csv', index=False)
df_test_result
```

	발생지_시군구	Risk	응급실_개수	시군구별_인구
0	양산시	3.829265	3	345978
1	유성구	1.067327	1	349197
2	울주군	2.003484	1	221909
3	강릉시	1.098438	4	213450
4	춘천시	8.475026	2	280815
...	...	...	...	...
4218	달서구	3.114486	6	573456
4219	하남시	1.073994	0	253119
4220	하남시	1.067327	0	253119
4221	화성시	4.979022	4	755068
4222	화성시	1.604285	4	755068

4223 rows × 4 columns

In [142]:

```
pred_top20 = df_test_result.groupby('발생지_시군구').mean().sort_values(by='Risk', ascending=False)[:20]
sb.barplot(pred_top20.index, [i[0] for i in pred_top20.values], palette='rainbow', alpha=.5)
plt.xticks(rotation = 90)
plt.xlabel('시군구')
plt.ylabel('Risk_mean')
plt.show()
```



Processing math: 100%

In [6]:

```

import os
import json
import folium
import pandas as pd
from folium import plugins
from folium.plugins import HeatMap, MarkerCluster
from sklearn.preprocessing import MinMaxScaler
from IPython.display import Image

import warnings
warnings.filterwarnings('ignore')

```

In [2]:

```

rfile = '시군구.geojson'
geo_str = json.load(open(rfile, encoding='utf-8'))
train_df = pd.read_csv('train1.csv')
test_df = pd.read_csv('test1.csv')
train_df = train_df.groupby('발생지_시군구').mean().sort_values(by='Risk', ascending=False)
test_df = test_df.groupby('발생지_시군구').mean().sort_values(by='Risk', ascending=False)

```

## 표준화 작업

- Risk, 응급실\_개수, 시군구별\_인구의 값은 모두 0이상의 값을 가지고 있으므로 0 ~ 1사이의 값으로 일괄적으로 표준화해주기 위해 Min-Max 이용.
- Train(실제), Test(예측) 각 데이터에 대하여 실시.

In [3]:

```

scaler = MinMaxScaler()
scaled_train = scaler.fit_transform(train_df)
df_scaled_train = pd.DataFrame(scaled_train)
df_scaled_train.columns = ['Risk', '응급실_개수', '시군구별_인구']
df_scaled_train.index = train_df.index
df_scaled_train

```

Out[3]:

Risk 응급실\_개수 시군구별\_인구

발생지_시군구	Risk	응급실_개수	시군구별_인구
무주군	1.000000	0.000000	0.020482
영광군	0.820513	0.111111	0.045231
통영시	0.758974	0.000000	0.111431
장수군	0.738462	0.000000	0.019083
장흥군	0.707692	0.055556	0.032797
...	...	...	...
연수구	0.029150	0.055556	0.291266
해운대구	0.000000	0.111111	0.341134
성북구	0.000000	0.055556	0.367559
목포시	0.000000	0.277778	0.193584
화천군	0.000000	0.000000	0.021128

155 rows × 3 columns

## Risk, 응급실수, 인구수를 고려한 시각화

In [4]:

```

scaler = MinMaxScaler()
scaled_test = scaler.fit_transform(test_df)
df_scaled_test = pd.DataFrame(scaled_test)
df_scaled_test.columns = ['Risk', '응급실_개수', '시군구별_인구']
df_scaled_test.index = test_df.index
df_scaled_test

```

Out[4]:

Risk 응급실\_개수 시군구별\_인구

발생지_시군구	Risk	응급실_개수	시군구별_인구
삼척시	1.000000	0.055556	0.056713
부여군	0.872868	0.000000	0.056667
단양군	0.811276	0.000000	0.025075
영주시	0.745144	0.055556	0.088998
남원시	0.665475	0.055556	0.068721
...	...	...	...

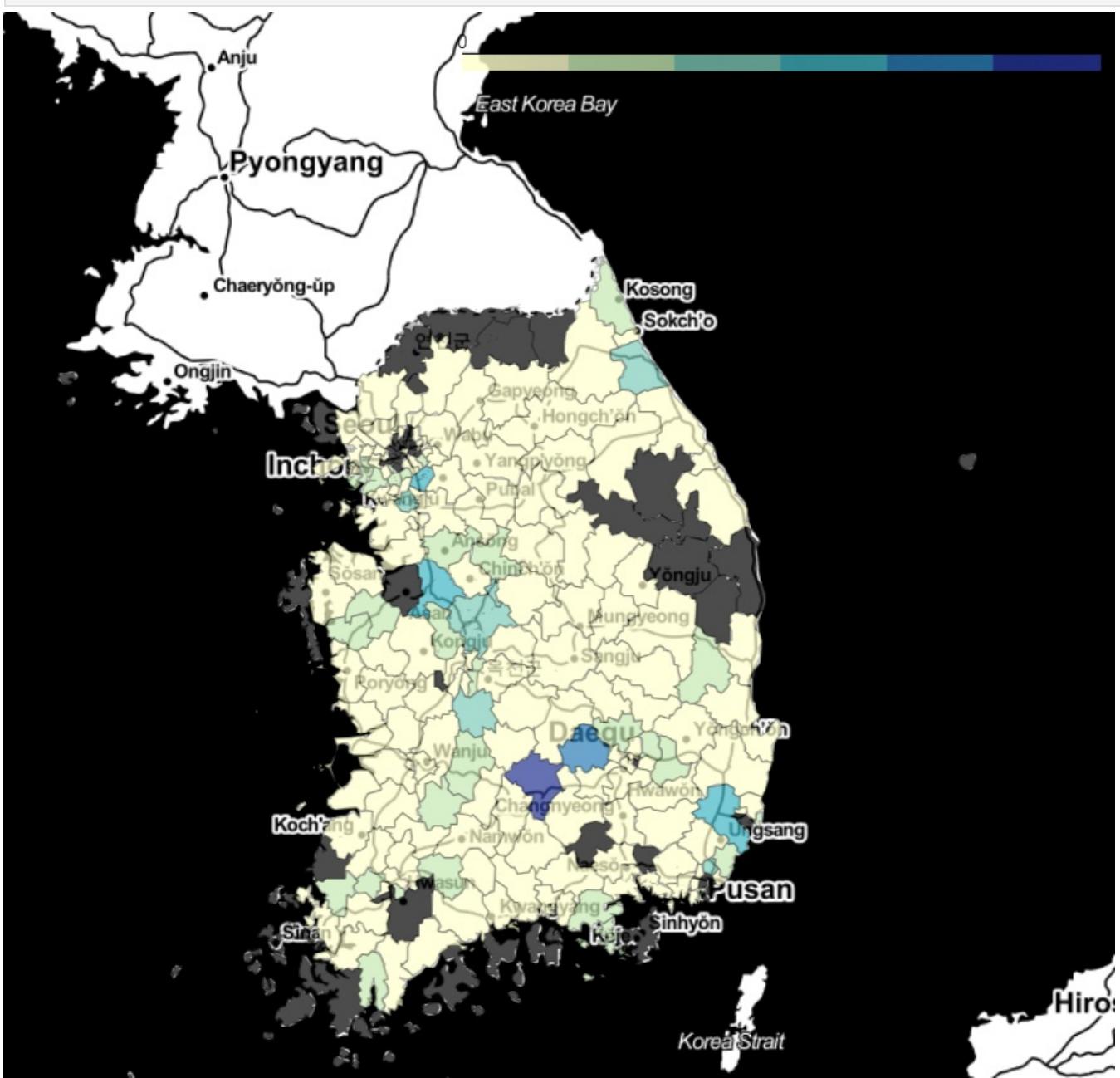
해운대구	0.039216	0.111111	0.341134
사상구	0.019866	0.111111	0.186388
영덕군	0.010361	0.000000	0.031681
강진군	0.000844	0.000000	0.030097
파주시	0.000000	0.055556	0.373440

155 rows × 3 columns

```
In [ ]: m_true = folium.Map(location=[36.8, 127.5], tiles= 'Stamen Toner',zoom_start = 7, width = 850)
m_true.choropleth(
    geo_data      = geo_str,
    data          = train_df,
    columns       = [test_df.index, '응급실_개수','Risk'],
    key_on        = 'properties.SIG_KOR_NM',
    fill_color    = 'YlGnBu',
    fill_opacity  = 0.7,
    line_opacity  = 0.3,
    colors=['red','blue']
)
# m_true.save('MAP_TRUE.html')
# m_true
```

In [11]: Image( '과정/1.png' )

Out[11]:



```
In [ ]: m_pred = folium.Map(location=[36.8, 127.5], tiles= 'Stamen Toner',zoom_start = 7, width = 850)
```

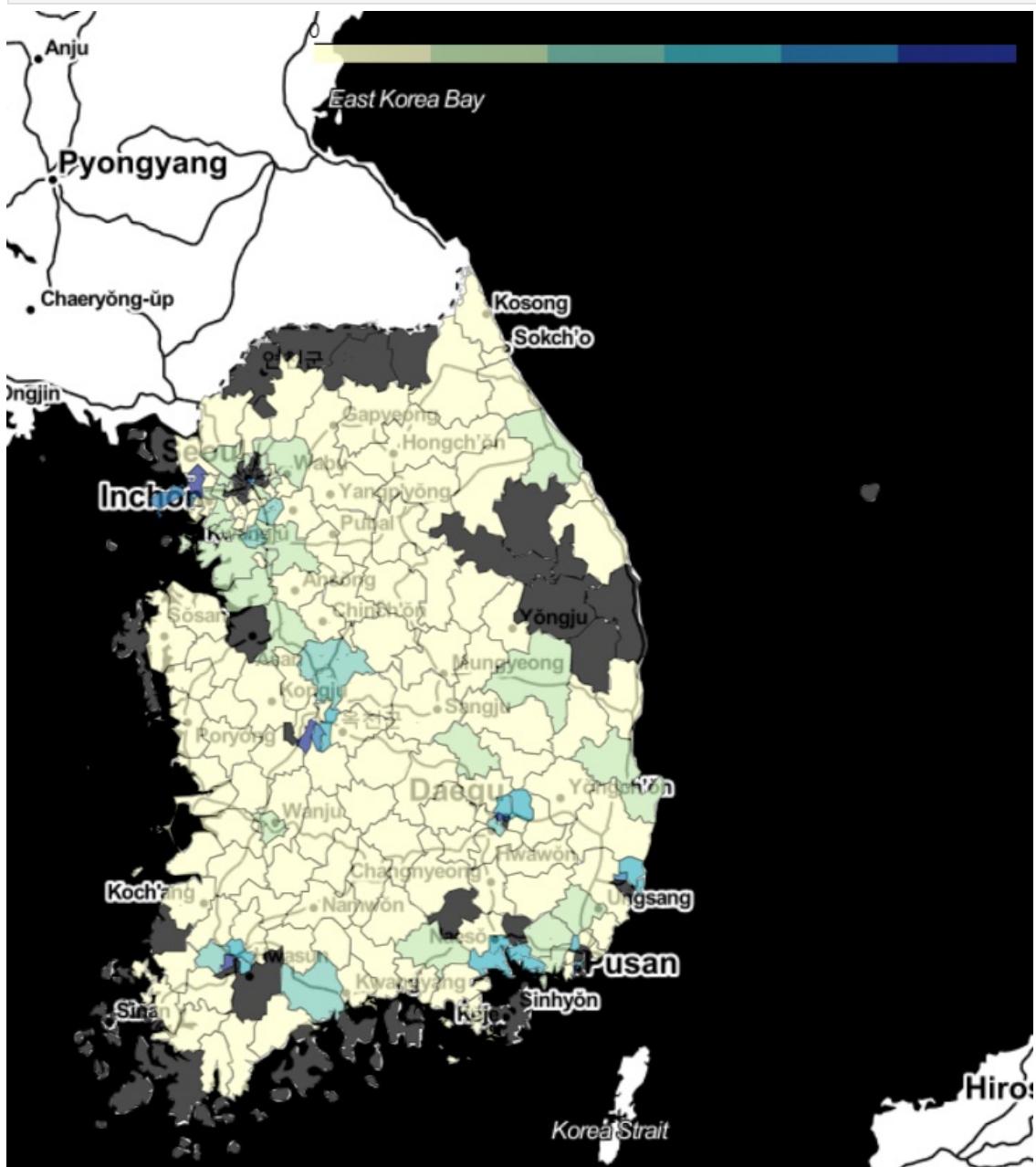
```

m_pred.choropleth(
    geo_data      = geo_str,
    data          = test_df,
    columns       = [test_df.index, '응급실 개수', 'Risk'],
    key_on        = 'properties.SIG_KOR_NM',
    fill_color    = 'YlGnBu',
    fill_opacity  = 0.7,
    line_opacity  = 0.3,
    colors=['red','blue']
)
# m_pred.save('MAP_PRED.html')
# m_pred

```

In [10]: `Image('과정/2.png')`

Out[10]:



In [ ]: `print(f" 전체 시군구 수 : {len(geo_str['features'])}개")`

Risk, 응급실수, 인구수를 고려한 서클 시각화 (Train Data = 실제값)

In [ ]:

```

def center_calc(points_df) :
    x = points_df.x
    y = points_df.y
    X = (max(x) + min(x)) / 2.
    Y = (max(y) + min(y)) / 2.
    return X, Y

def points_array(points) :
    final_points = []
    for x in range(0, len(points)) :

```

```

    if len(points[x]) == 2 :
        final_points.append(points[x])
    else :
        target = points[x]
        for y in range(0, len(target)) :
            final_points.append(target[y])
return final_points

center_locations = pd.DataFrame()
codes = []
names = []
x_list = []
y_list = []

for x in range(0, len(geo_str['features'])):
    code = geo_str['features'][x]['properties']['SIG_CD']
    name = geo_str['features'][x]['properties']['SIG_KOR_NM']

    points = geo_str['features'][x]['geometry']['coordinates'][0]
    points = points_array(points)
    points_df = pd.DataFrame(points)
    points_df.columns = ['x','y']
    X, Y = center_calc(points_df)

    codes.append(code)
    names.append(name)
    x_list.append(X)
    y_list.append(Y)

# print(len(codes), len(names), len(x_list),len(y_list))

center_locations['CODE'] = codes
center_locations['NAME'] = names
center_locations['X'] = x_list
center_locations['Y'] = y_list

df = df_scaled_train
inter = set(df.index) & set(center_locations['NAME'])
code_dic = { name : code for name, code in zip(center_locations['NAME'], center_locations['CODE'])} if name in inter
df['CODE'] = [code_dic.get(i) for i in df.index]

center_locations['X'] = center_locations['X'].astype('str')
center_locations['Y'] = center_locations['Y'].astype('str')
center_locations['XY'] = center_locations['X'] + ',' + center_locations['Y']
lat_lon = {code : xy for code, xy in zip(center_locations['CODE'], center_locations['XY'])}

df_lat_lon = [lat_lon.get(i) for i in df['CODE']]

l1, l2 = [], []
for i in df_lat_lon :
    li = i.split(',')
    lat = li[0]
    lon = li[-1]
    l1.append(lat)
    l2.append(lon)

df['lat'] = l1
df['lon'] = l2

m = folium.Map(location = [36.8, 127.5], tiles = 'Stamen Toner', zoom_start = 7, width = 1000)

m.choropleth(
    geo_data = geo_str,
    data = df,
    columns = [test_df.index, '응급실_개수','Risk', '시군구별_인구'],
    key_on = 'properties.SIG_KOR_NM',
    fill_color = 'PuRd',
    fill_opacity = 0.7,
    line_opacity = 0.3,
    colors=['red','blue']
)

for idx in range(0, len(df)) :
    latitude = df['lon'][idx]
    longitude = df['lat'][idx]
    location = (latitude, longitude)
    folium.CircleMarker(location, radius = df['Risk'][idx] * 20,
                        popup = df.index[idx],
                        fill_color='#3186cc',
                        color='blue').add_to(m)

folium.LayerControl(collapsed=False).add_to(m)

# -----
for idx in range(0, len(df)) :
    latitude = df['lon'][idx]
    longitude = df['lat'][idx]
    location = (latitude, longitude)

```

```

        folium.CircleMarker(location, radius = df['응급실_개수'][idx] * 20,
                             popup = df.index[idx],
                             color='red').add_to(m)

folium.LayerControl(collapsed=False).add_to(m)

# -----
for idx in range(0, len(df)) :
    latitude = df['lon'][idx]
    longitude = df['lat'][idx]
    location = (latitude, longitude)
    folium.CircleMarker(location, radius = df['시군구별_인구'][idx] * 20,
                         popup = df.index[idx],
                         color='yellow').add_to(m)

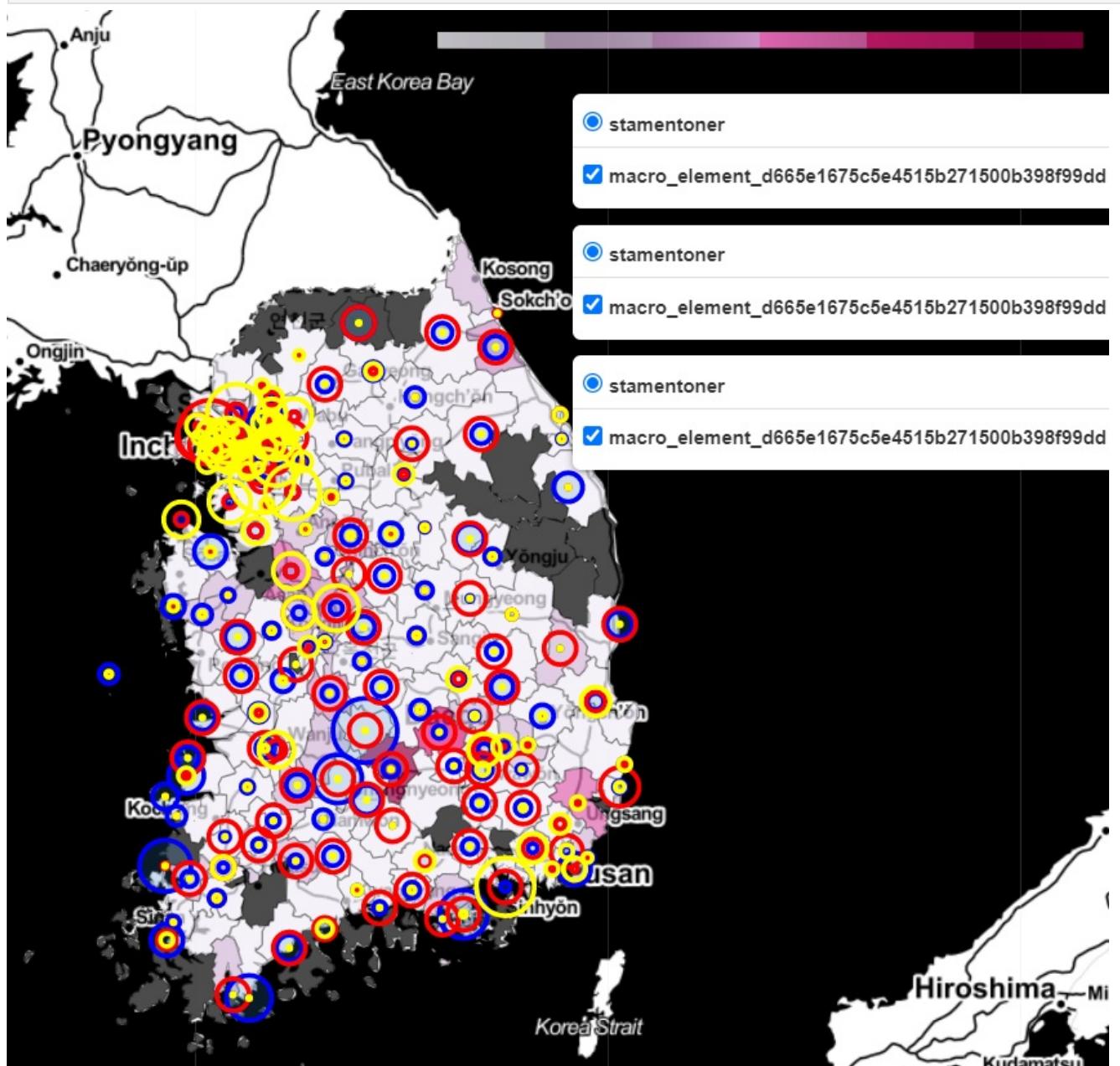
folium.LayerControl(collapsed=False).add_to(m)

m.save('MAP_RISK.html')
m

```

In [12]: Image('과정/3.png')

Out[12]:



Risk + 응급실 수 + 인구수의 총합을 고려한 순위 생성 및 시각화(Train Data = 실제값)

In [ ]:

```

rank = -(-df_scaled_train.iloc[:, :1].values + df_scaled_train.iloc[:, 1:2].values + df_scaled_train.iloc[:, 2:3])
df['rank_risk'] = rank
df_top_10 = df.sort_values("rank_risk", ascending = False)[:10]

rank_m = folium.Map(location = [36.8, 127.5], tiles = 'Stamen Toner', zoom_start = 7, width = 1000)

df = df_top_10

```

```

rank_m.choropleth(
    geo_data      = geo_str,
    data          = df,
    columns       = [df.index, 'rank_risk'],
    key_on        = 'properties.SIG_KOR_NM',
    fill_color    = 'PuRd',
    fill_opacity  = 0.7,
    line_opacity  = 0.3,
    colors=['red']
)

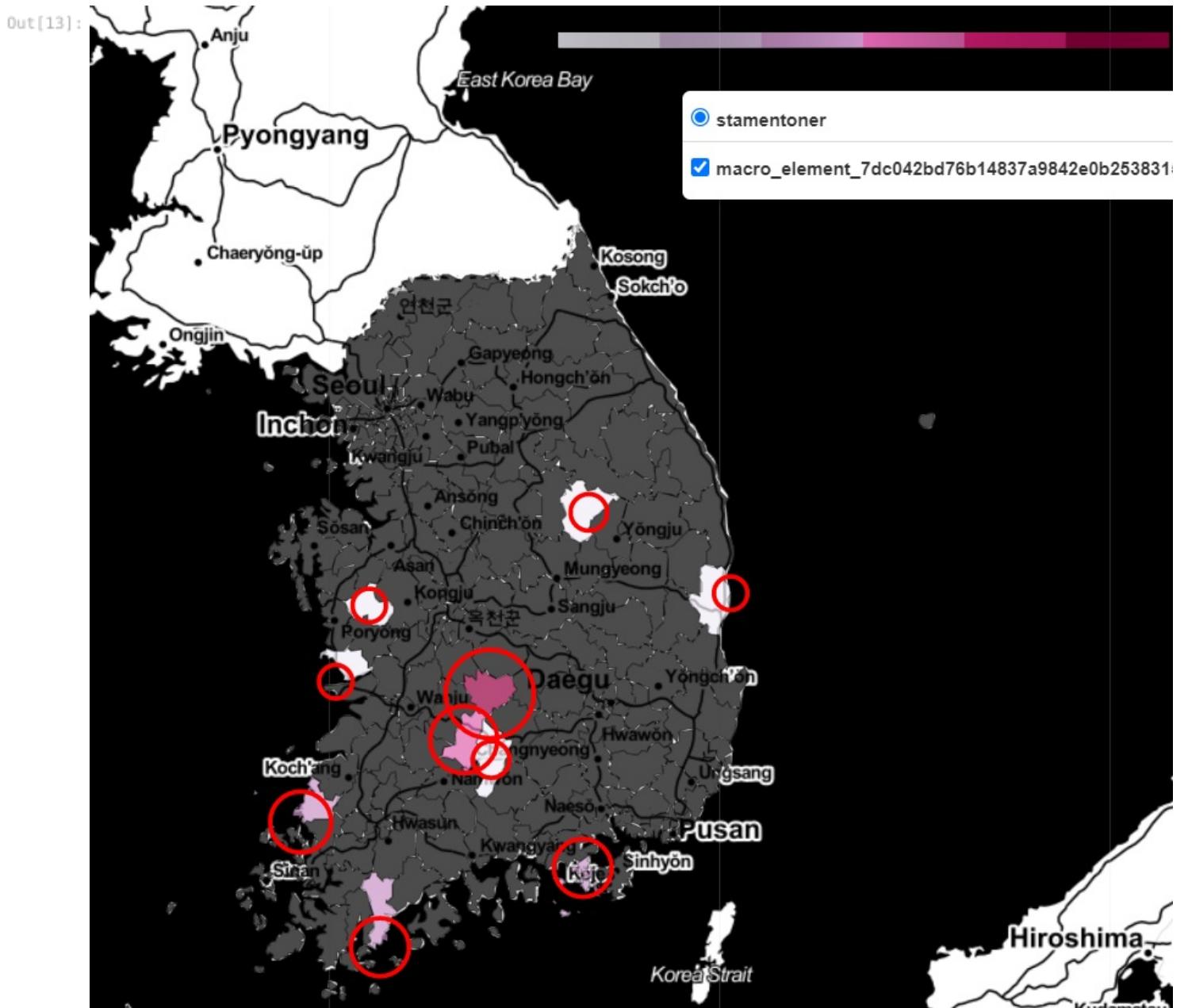
for idx in range(0, len(df)) :
    latitude  = df['lon'][idx]
    longitude = df['lat'][idx]
    location  = (latitude, longitude)
    folium.CircleMarker(location, radius = df['rank_risk'][idx] * 30,
                         popup = df.index[idx],
                         color ='red').add_to(rank_m)

folium.LayerControl(collapsed=False).add_to(rank_m)

rank_m.save('MAP_RANK.html')
rank_m

```

In [13]: Image('과정/5.png')



QGIS를 이용하기 위한 상위 10개 위험도 분석 지역 데이터 저장(실제)

In [ ]:

```

df_top_10.to_csv('true.csv', encoding = 'euc-kr')
df_top_10

```

Risk + 응급실 수 + 인구수의 총합을 고려한 순위 생성 및 시각화(Test Data = 예측값)

```
In [ ]:  
def center_calc(points_df) :  
    x = points_df.x  
    y = points_df.y  
    X = (max(x) + min(x)) / 2.  
    Y = (max(y) + min(y)) / 2.  
    return X, Y  
  
def points_array(points) :  
    final_points = []  
    for x in range(0, len(points)) :  
        if len(points[x]) == 2 :  
            final_points.append(points[x])  
        else :  
            target = points[x]  
            for y in range(0, len(target)) :  
                final_points.append(target[y])  
    return final_points  
  
center_locations = pd.DataFrame()  
codes = []  
names = []  
x_list = []  
y_list = []  
  
for x in range(0, len(geo_str['features'])):  
    code = geo_str['features'][x]['properties']['SIG_CD']  
    name = geo_str['features'][x]['properties']['SIG_KOR_NM']  
  
    points = geo_str['features'][x]['geometry']['coordinates'][0]  
    points = points_array(points)  
    points_df = pd.DataFrame(points)  
    points_df.columns = ['x','y']  
    X, Y = center_calc(points_df)  
  
    codes.append(code)  
    names.append(name)  
    x_list.append(X)  
    y_list.append(Y)  
  
# print(len(codes), len(names), len(x_list),len(y_list))  
  
center_locations['CODE'] = codes  
center_locations['NAME'] = names  
center_locations['X'] = x_list  
center_locations['Y'] = y_list  
  
df = df_scaled_test  
inter = set(df.index) & set(center_locations['NAME'])  
code_dic = { name : code for name, code in zip(center_locations['NAME'], center_locations['CODE']) } if name in inter : df['CODE'] = [code_dic.get(i) for i in df.index]  
  
center_locations['X'] = center_locations['X'].astype('str')  
center_locations['Y'] = center_locations['Y'].astype('str')  
center_locations['XY'] = center_locations['X'] + ',' + center_locations['Y']  
lat_lon = {code : xy for code, xy in zip(center_locations['CODE'], center_locations['XY'])}  
  
df_lat_lon = [lat_lon.get(i) for i in df['CODE']]  
  
l1, l2 = [], []  
for i in df_lat_lon :  
    li = i.split(',')  
    lat = li[0]  
    lon = li[-1]  
    l1.append(lat)  
    l2.append(lon)  
  
df['lat'] = l1  
df['lon'] = l2  
  
m = folium.Map(location = [36.8, 127.5], tiles = 'Stamen Toner', zoom_start = 7, width = 1000)  
m.choropleth(  
    geo_data = geo_str,  
    data = df,  
    columns = [test_df.index, '응급실_개수', 'Risk', '시군구별_인구'],  
    key_on = 'properties.SIG_KOR_NM',  
    fill_color = 'PuRd',  
    fill_opacity = 0.7,  
    line_opacity = 0.3,  
    colors=['red','blue'])
```

```

for idx in range(0, len(df)) :
    latitude = df['lon'][idx]
    longitude = df['lat'][idx]
    location = (latitude, longitude)
    folium.CircleMarker(location, radius = df['Risk'][idx] * 20,
                        popup = df.index[idx],
                        fill_color='#3186cc',
                        color='blue').add_to(m)

folium.LayerControl(collapsed=False).add_to(m)

# -----
for idx in range(0, len(df)) :
    latitude = df['lon'][idx]
    longitude = df['lat'][idx]
    location = (latitude, longitude)
    folium.CircleMarker(location, radius = df['응급실_개수'][idx] * 20,
                        popup = df.index[idx],
                        color='red').add_to(m)

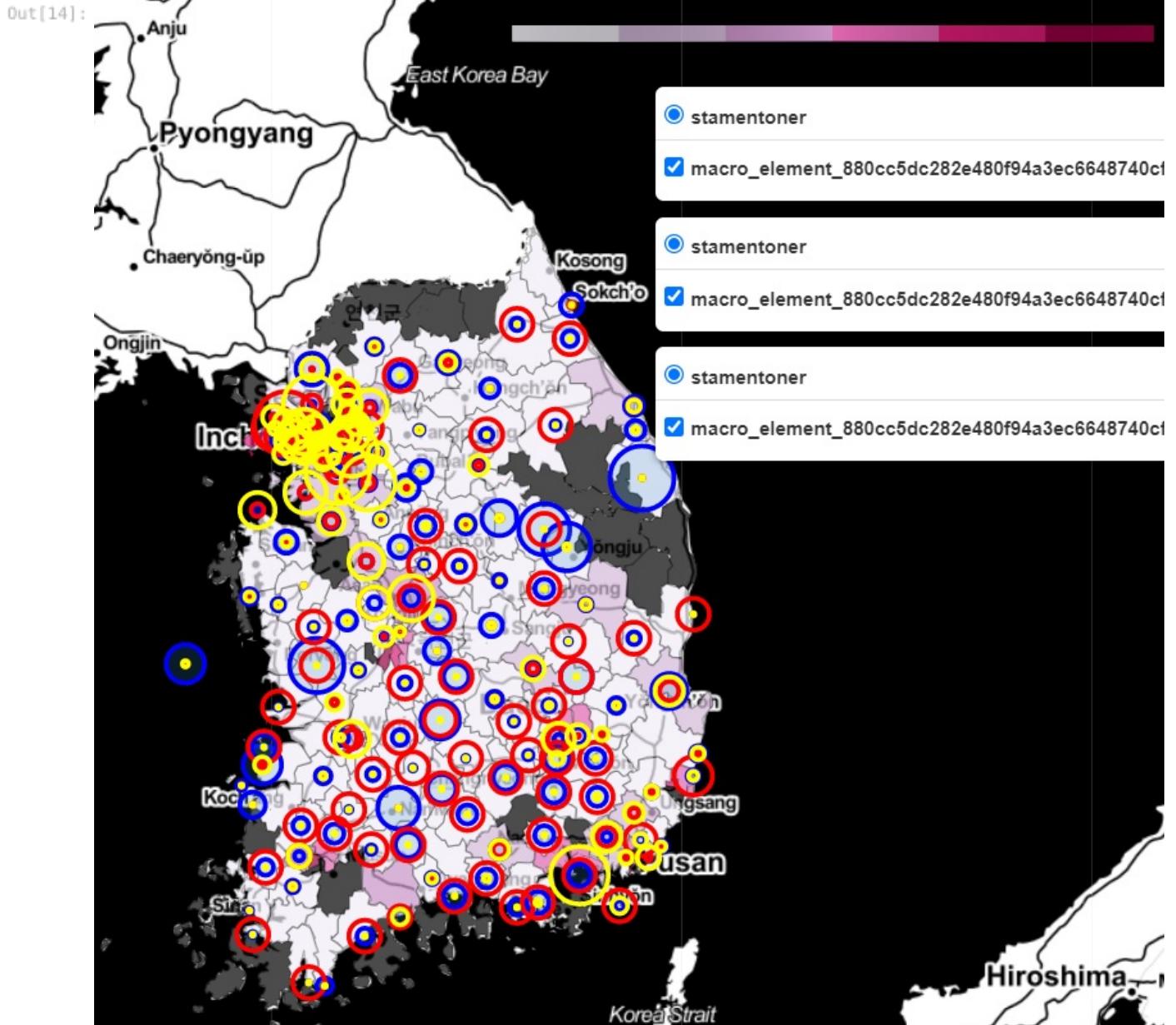
folium.LayerControl(collapsed=False).add_to(m)

# -----
for idx in range(0, len(df)) :
    latitude = df['lon'][idx]
    longitude = df['lat'][idx]
    location = (latitude, longitude)
    folium.CircleMarker(location, radius = df['시군구별_인구'][idx] * 20,
                        popup = df.index[idx],
                        color='yellow').add_to(m)

folium.LayerControl(collapsed=False).add_to(m)
m

```

In [14]: Image('과정/4.png')



Risk, 응급실수, 인구수를 고려한 서클 시각화 (Test Data = 예측값)

```
In [1]: rank = -(-df_scaled_test.iloc[:, :1].values + df_scaled_test.iloc[:, 1:2].values + df_scaled_test.iloc[:, 2:3].values)
df['rank_risk'] = rank
df_top_10 = df.sort_values("rank_risk", ascending = False)[:10]

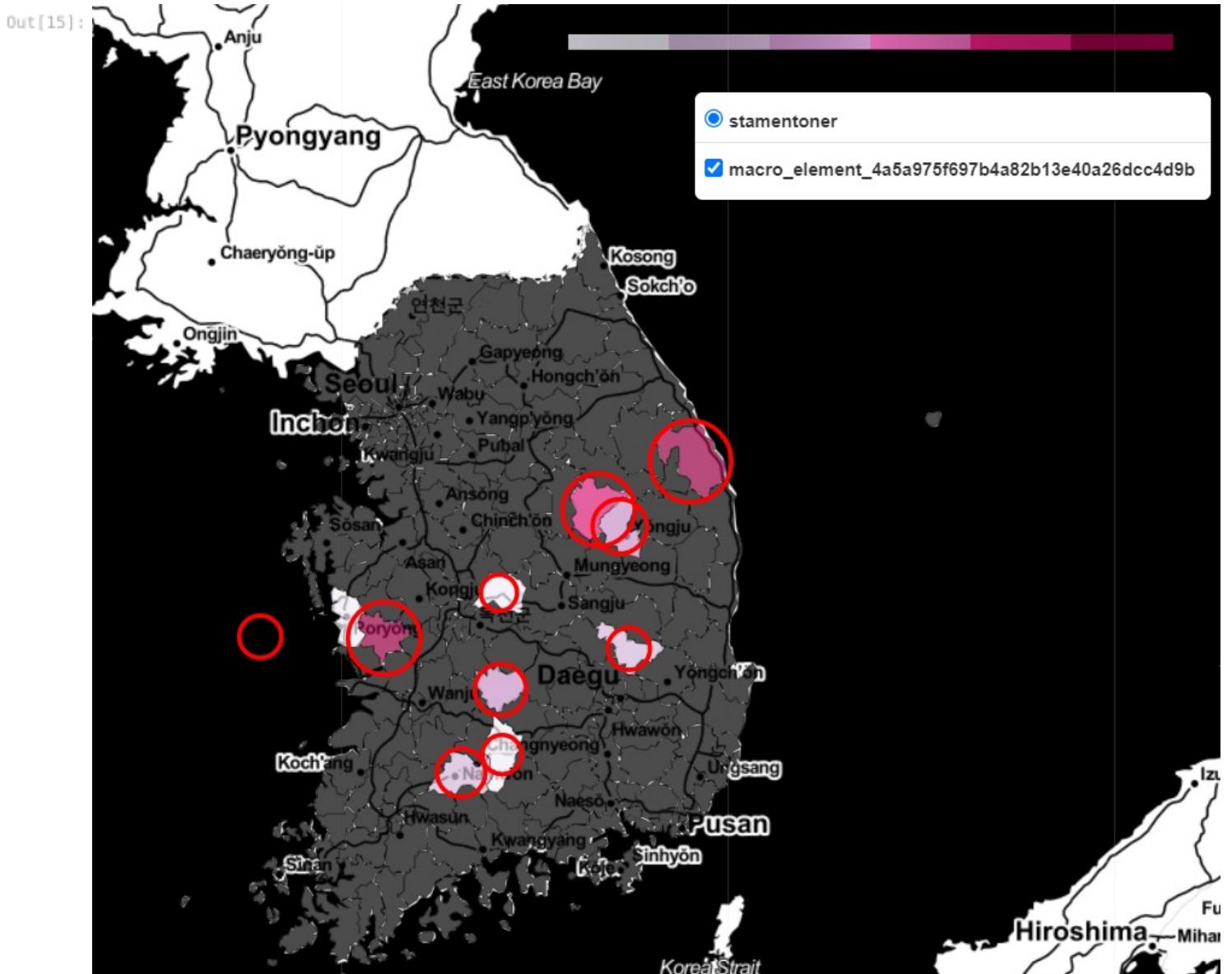
rank_m = folium.Map(location = [36.8, 127.5], tiles = 'Stamen Toner', zoom_start = 7, width = 1000)

df = df_top_10
rank_m.choropleth(
    geo_data      = geo_str,
    data          = df,
    columns       = [df.index, 'rank_risk'],
    key_on        = 'properties.SIG_KOR_NM',
    fill_color    = 'PuRd',
    fill_opacity  = 0.7,
    line_opacity  = 0.3,
    colors=['red']
)

for idx in range(0, len(df)) :
    latitude  = df['lon'][idx]
    longitude = df['lat'][idx]
    location  = (latitude, longitude)
    folium.CircleMarker(location, radius = df['rank_risk'][idx] * 30,
                         popup = df.index[idx],
                         color = 'red').add_to(rank_m)

folium.LayerControl(collapsed=False).add_to(rank_m)
rank_m
```

In [15]: Image('과정/6.png')



QGIS를 이용하기 위한 상위 10개 위험도 분석 지역 데이터 저장(예측)

```
In [ ]: df_top_10.to_csv('pred.csv', encoding = 'euc-kr')  
df_top_10
```

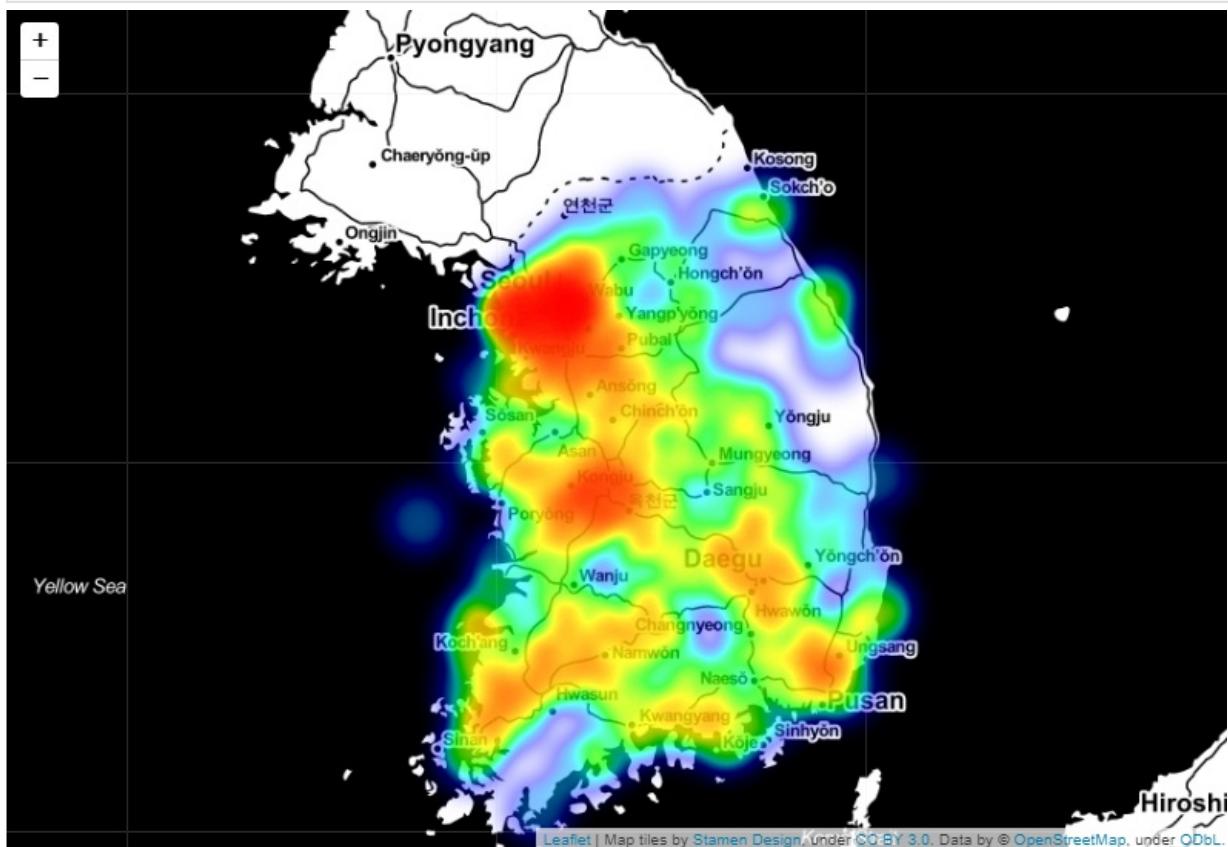
참고

- Heatmap

```
In [ ]: m = folium.Map(location = [36.8, 127.5], tiles = 'Stamen Toner', zoom_start = 7, width = 850)  
heat_df = df[['lat', 'lon']]  
heat_data = [[row['lon'], row['lat']] for index, row in heat_df.iterrows()]  
HeatMap(heat_data).add_to(m)  
m
```

```
In [16]: Image('과정/7.png')
```

Out[16]:



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js