머신러닝으로 교통사고 예측하기

꿔8로우팀 박지수

서울시 교통사고 사망 데이터를 도로 형태별로 시각화하기

사용 데이터 : 도로교통공단_사망사고_개별정보(2012~2020년)

17~19년도의 서울특별시 교통사고 데이터로 분석하기 위해 조건을 만족하는 데이터만 뽑아내었으며,
발생년월일시 데이터를 datetime 자료형에 맞게 변환하는 작업과,
발생월, 일, 시를 datetime의 메소드를 사용하여 분해하는 작업을 거쳤습니다.

```
death.head()
```

	발생년	발생년월일시	주야	요일	사망자수	사상자수	중상자수	경상자수	부상신고자수	발생지시도	발생지시군구	사고유형_대분류	사고유형_중분류	사고유형	법규위반	도로형태_대분류	도로형태	당사자종별_1당_대분류	당사자종별_2당_대분류	발생위치X_UTMK	발생위치Y_UTMK	경도	위도
0	2012	2012010101	야간	일	1	1	0	0	0	서울	은평구	차대사람	차도통행중	차도통행중	안전운전의무 불이행	단일로	기타단일로	승용차	보행자	949860	1957179	126.931891	37.612680
1	2012	2012010101	야간	일	1	6	5	0	0	전북	정읍시	차대차	정면충돌	정면충돌	중앙선 침범	단일로	기타단일로	승용차	승용차	946537	1737695	126.909523	35.633956
2	2012	2012010108	주간	일	1	1	0	0	0	충남	청양군	차량단독	공작물충돌	공작물충돌	안전운전의무 불이행	단일로	기타단일로	승용차	없음	940016	1832833	126.830281	36.491268
3	2012	2012010110	주간	일	2	2	0	0	0	경남	합천군	차대차	측면충돌	측면충돌	과속	교차로	교차로내	승합차	승용차	1059321	1748774	128.155984	35.733503
4	2012	2012010103	야간	일	1	1	0	0	0	경북	예천군	차량단독	도로이탈	도로이탈 추락	안전운전의무 불이행	단일로	기타단일로	승용차	없음	1070222	1834630	128.284180	36.506769

서울시 교통사고 사망 데이터를 도로 형태별로 시각화하기

사용 데이터 : 도로교통공단_사망사고_개별정보(2012~2020년)

17~19년도의 서울특별시 교통사고 데이터로 분석하기 위해 조건을 만족하는 데이터만 뽑아내었으며,
발생년월일시 데이터를 datetime 자료형에 맞게 변환하는 작업과,
발생월, 일, 시를 datetime의 메소드를 사용하여 분해하는 작업을 거쳤습니다.

```
[ ]  death = pd.read_csv('/content/drive/MyDrive/꿔8로우/12_20_death.csv', encoding='cp949')
```

```
[ ]  death = death[(death['발생년'] >=2017) & (death['발생년'] <= 2019)].copy()

[ ]  death_seoul = death[death['발생지시도'] == '서울'].copy()

[ ]  death_seoul['발생년월일시'] = pd.to_datetime(death_seoul['발생년월일시'], format='%Y%m%d%H')

[ ]  death_seoul['발생월'], death_seoul['발생일'], death_seoul['발생시'] = (death_seoul['발생년월일시'].dt.month, death_seoul['발생년월일시'].dt.day, death_seoul['발생년월일시'].dt.hour)

[ ]  death_seoul = death_seoul.sort_values('발생년월일시', ascending=True)
     death_seoul
```

서울시 교통사고 사망 데이터를 도로 형태별로 시각화하기

발생년월일시에 맞게 오름차순으로 교통사고 사망 데이터를 정렬하면,
다음과 같이 정렬되며, 새로 추가한 열이 맨 오른쪽에 생기는 것을 확인할 수 있습니다.

```
death_seoul = death_seoul.sort_values('발생년월일시', ascending=True)
death_seoul.head()
```

	발생년	발생년월일시	주야	요일	사망자수	사상자수	중상자수	경상자수	부상신고자수	발생지시도	발생지시군구	사고유형_대분류	사고유형_중분류	사고유형	법규위반	도로형태_대분류	도로형태	당사자종별_1당_대분류	당사자종별_2당_대분류	발생위치X_UTMK	발생위치Y_UTMK	경도	위도	발생월	발생일	발생시
23193	2017	2017-01-01 02:00:00	야간	일	1	1	0	0	0	서울	송파구	차대사람	횡단중	횡단중	안전운전의무 불이행	교차로	교차로부근	승용차	보행자	967570	1944450	127.133107	37.498741	1	1	2
23194	2017	2017-01-01 04:00:00	야간	일	1	1	0	0	0	서울	금천구	차대사람	기타	기타	안전운전의무 불이행	단일로	기타단일로	승용차	보행자	946778	1941695	126.898094	37.472946	1	1	4
23202	2017	2017-01-02 04:00:00	야간	월	1	1	0	0	0	서울	은평구	차대사람	횡단중	횡단중	보행자보호의무 위반	교차로	교차로내	승용차	보행자	948413	1957900	126.915444	37.619097	1	2	4
23206	2017	2017-01-02 08:00:00	주간	월	1	1	0	0	0	서울	영등포구	차량단독	전도전복	전도전복	안전운전의무 불이행	교차로	교차로내	이륜차	없음	946757	1943309	126.897739	37.487492	1	2	8
23211	2017	2017-01-02 17:00:00	주간	월	1	5	0	4	0	서울	금천구	차대차	측면충돌	측면충돌	신호위반	교차로	교차로내	승용차	승합차	947235	1938476	126.903492	37.443959	1	2	17

서울시 교통사고 사망 데이터를 도로 형태별로 시각화하기

이제, 서울특별시 교통사고 사망 데이터를 도로형태와 위도, 경도로만 이루어진 부분 데이터셋을 만들어주는 작업을 거치겠습니다.

결측값이 있는지 확인해주는 작업을 하기 위해 isna().sum() 메서드를 적용시켜 주었습니다.

```
death = death_seoul[['도로형태','위도', '경도']]
death
```

	도로형태	위도	경도
23193	교차로부근	37.498741	127.133107
23194	기타단일로	37.472946	126.898094
23202	교차로내	37.619097	126.915444
23206	교차로내	37.487492	126.897739
23211	교차로내	37.443959	126.903492
...	...	...	...
34099	교차로내	37.628761	126.919471
34127	기타단일로	37.590333	127.062885
34128	기타단일로	37.493253	126.926969
34139	기타단일로	37.500788	126.983592
34140	교차로부근	37.485819	127.097769

885 rows × 3 columns

```
death_seoul.isna().sum()
```

```
발생년                0
발생년월일시              0
주야                 0
요일                 0
사망자수                0
사상자수                0
중상자수                0
경상자수                0
부상신고자수              0
발생지시도               0
발생지시군구              0
사고유형_대분류            0
사고유형_중분류            0
사고유형                0
법규위반                0
도로형태_대분류            0
도로형태                0
당사자종별_1당_대분류       0
당사자종별_2당_대분류       0
발생위치X_UTMK         0
발생위치Y_UTMK         0
경도                 0
위도                 0
발생월                 0
발생일                 0
발생시                 0
dtype: int64
```

서울시 교통사고 사망 데이터를 도로 형태별로 시각화하기

Folium 모듈을 임포트해와서, Map 메소드를 사용하여 지도 위에 시각화하는 작업을 거치겠습니다.
'도로형태' 별로 각각의 교통사고 사망 데이터를 다른 색으로 표시할 수 있도록 설정하는
수작업을 추가하였습니다.

```
import folium
```

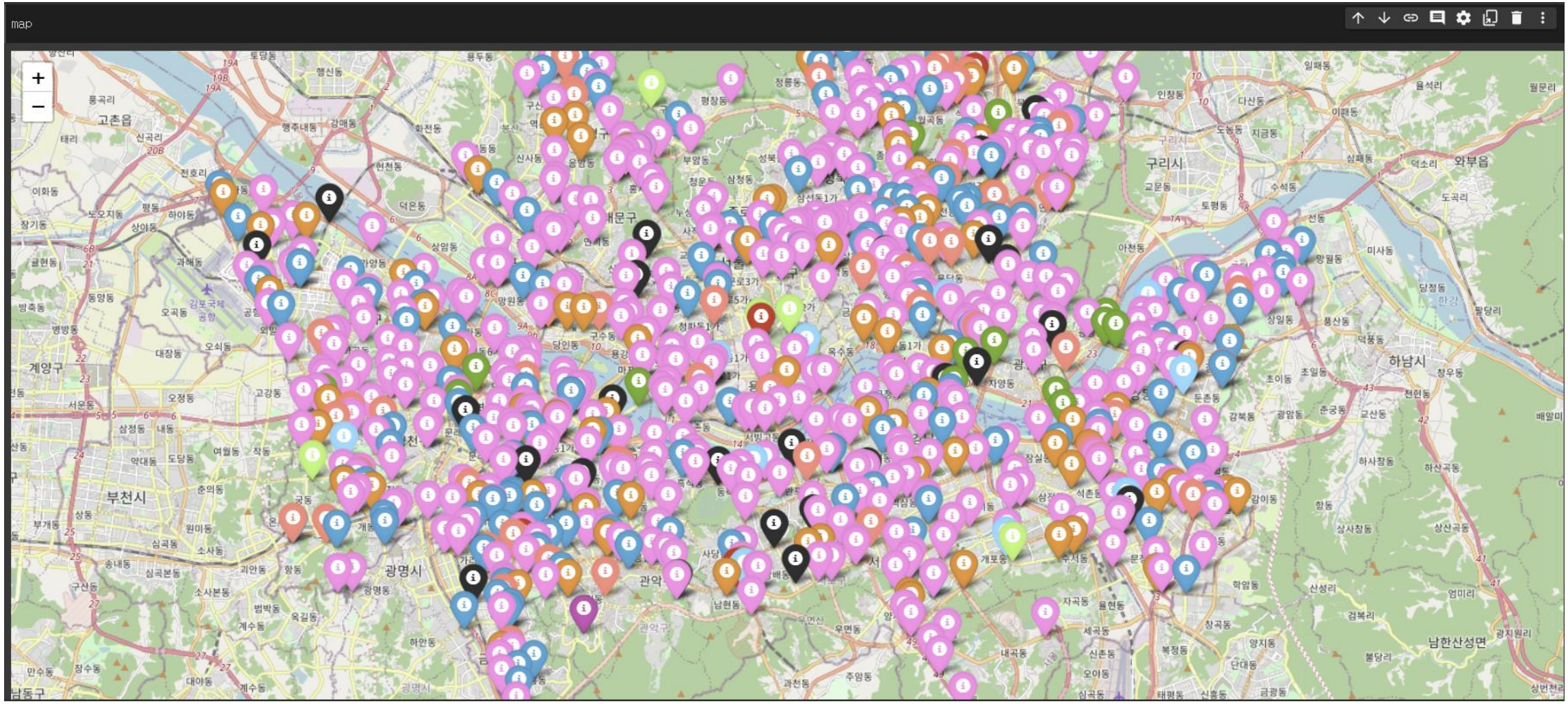
```
import folium
from folium import Marker, Icon
map = folium.Map(location=[37.5, 127], zoom_start=12) # 지도의 확대범위(zoom_start)
```

```
death['도로형태'].unique()
```

```
array(['교차로부근', '기타단일로', '교차로내', '지하차도(도로)내', '기타', '고가도로위', '교차로횡단보도내',
       '교량위', '터널안', '주차장', '불명'], dtype=object)
```

```
for i in death.index:
    shape = death.loc[i, '도로형태']
    lat = death.loc[i, '위도']
    lon = death.loc[i, '경도']
    if death['도로형태'][i] == '고가도로위':
        code_color = 'red'
    elif death['도로형태'][i] == '교량위':
        code_color = 'green'
    elif death['도로형태'][i] == '교차로내':
        code_color = 'blue'
    elif death['도로형태'][i] == '교차로부근':
        code_color = 'orange'
    elif death['도로형태'][i] == '교차로횡단보도내':
        code_color = 'lightred'
    elif death['도로형태'][i] == '주차장':
        code_color = 'purple'
    elif death['도로형태'][i] == '지하차도(도로)내':
        code_color = 'lightblue'
    elif death['도로형태'][i] == '터널안':
        code_color = 'lightgreen'
    elif death['도로형태'][i] == '기타단일로':
        code_color = 'pink'
    else : code_color = 'black'
    marker = folium.Marker([lat,lon], popup=shape, icon=folium.Icon(color=code_color)).add_to(map)
```

서울시 교통사고 사망 데이터를 도로 형태별로 시각화하기



대부분 교통사고 사망 사건이 단일로(분홍색)에서 발생한 것을 알 수 있었으며,

구로구, 은평구, 동대문구, 송파구 등의 지역에서는 교차로내 혹은 부근(파란색, 주황색)에서 발생한 것을 확인할 수 있었습니다.

서울시 개별사고 데이터를 사용하여 연령대별 사고 유형 분석하기

사용 데이터 : 도로교통공단_서울시_교통사고_개별데이터(2010~2018년)

먼저 데이터프레임의 형태를 살펴보기 위해 head( )로 앞의 3행만 추출해보았습니다.

```
df = pd.read_csv('/content/drive/MyDrive/꿔8로우/data/서울시 개별사고정보(2010_2018).csv', sep=',', encoding='cp949')
```

```
# df의 앞 3행만 훑어보기!
df.head(3)
```

	발생일	발생시간	요일	발생지_시군구	사고내용	법정동명	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해성별	가해자연령	가해자신체상해정도	피해자차종	피해자성별	피해자연령	피해자신체상해정도
0	20100101	00시	금	마포구	경상	서교동	0	0	1	0	차대차	기타	기타	기타	포장	젖음/습기	맑음	단일로	기타단일로	승용차	남	54세	상해없음	이륜차	남	19세	경상
1	20100101	00시	금	동작구	중상	대방동	0	1	3	0	차대차	추돌	진행중 추돌	안전거리 미확보	포장	서리/결빙	맑음	단일로	기타단일로	승합차	남	57세	상해없음	승용차	남	39세	중상
2	20100101	00시	금	관악구	경상	신림동	0	0	2	0	차대차	추돌	진행중 추돌	기타	포장	서리/결빙	흐림	단일로	기타단일로	승용차	남	51세	경상	승용차	남	58세	경상

서울시 개별사고 데이터를 사용하여 연령대별 사고 유형 분석하기

데이터프레임의 칼럼 순서를 정렬하는 작업과, 가해자연령 및 피해자연령을 숫자 형으로 변환하는 작업을 거치겠습니다.

```
df = df.reindex(columns=['발생일','발생년도','발생월','발생일자','발생시간','요일','발생지_시군구','사고내용','법정동명','사망자수','중상자수','경
df.head(3)
```

```
df_subset = df[['발생년도','사고내용','사고유형','도로형태','가해자법규위반','가해자차종','피해자차종','가해자연령','피해자연령']]

df_subset['피해자연령'] = df_subset['피해자연령'].str.replace('세','')
df_subset['가해자연령'] = df_subset['가해자연령'].str.replace('세','')
df_subset['가해자연령'] = pd.to_numeric(df_subset['가해자연령'], errors='coerce')
df_subset['피해자연령'] = pd.to_numeric(df_subset['피해자연령'], errors='coerce')
df_subset
```

df_subset 완성본 ->

	발생년도	사고내용	사고유형	도로형태	가해자법규위반	가해자차종	피해자차종	가해자연령	피해자연령
0	2010	경상	기타	기타단일로	기타	승용차	이륜차	54.0	19.0
1	2010	중상	진행중 추돌	기타단일로	안전거리 미확보	승합차	승용차	57.0	39.0
2	2010	경상	진행중 추돌	기타단일로	기타	승용차	승용차	51.0	58.0
3	2010	경상	기타	교차로부근	안전거리 미확보	승용차	승용차	56.0	55.0
4	2010	경상	정면충돌	교차로내	신호위반	승용차	승용차	33.0	55.0
...	...	...	...	...	...	...	...	...	...
362292	2018	경상	기타	기타단일로	안전운전 의무 불이행	승용차	보행자	34.0	53.0
362293	2018	부상신고	횡단중	교차로횡단보도내	신호위반	승용차	보행자	56.0	35.0
362294	2018	경상	정면충돌	교차로내	신호위반	승용차	승용차	57.0	53.0
362295	2018	경상	길가장자리구역통행중	지하차도(도로)내	안전운전 의무 불이행	승용차	보행자	59.0	36.0
362296	2018	경상	기타	교차로부근	안전운전 의무 불이행	승용차	보행자	69.0	32.0

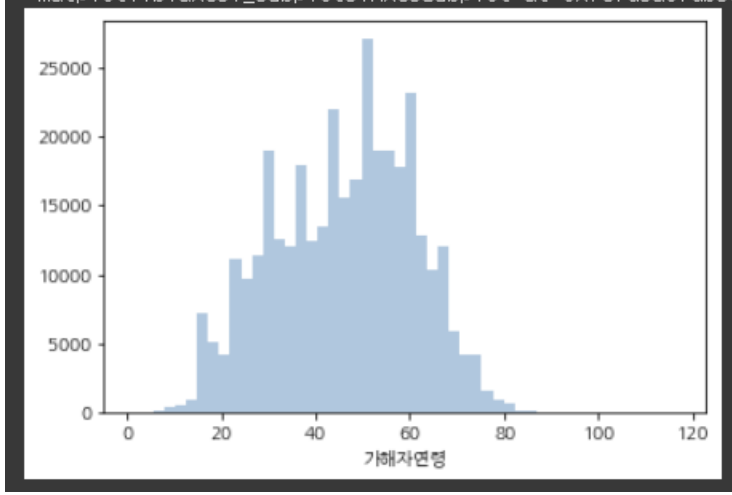
362297 rows × 9 columns

서울시 개별사고 데이터를 사용하여 연령대별 사고 유형 분석하기

가해자연령에 따른 서울시 개별사고의 분포를 시각화하기 위해, seaborn 라이브러리의 distplot 함수를 사용하였으며, 연령대 변수를 추가하기 위해, aging()이라는 연령대 함수를 만들었습니다.
또한, 결측값은 notna() 함수를 사용하여, 제외하는 작업을 거쳐주었습니다.

```
sns.distplot(df_subset['가해자연령'], kde=False)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7efd8a0fdb50>
```




```
# 연령대 함수 만들기
def aging(x):
  if x >= 60:
    return "60세 이상"
  elif x >= 40:
    return "40세 이상 59세 이하"
  elif x >= 19:
    return "19세 이상 39세 이하"
  else:
    return "18세 이하"
```

```
df_subset['피해자 연령대'] = '불명'
df_subset['가해자 연령대'] = '불명'
df_subset = df_subset[df_subset['가해자연령'].notna()] # 연령에서 결측값은 제외하도록 하겠습니다.
df_subset = df_subset[df_subset['피해자연령'].notna()]
df_subset['가해자 연령대'] = df_subset['가해자연령'].apply(aging)
df_subset['피해자 연령대'] = df_subset['피해자연령'].apply(aging)
```

서울시 개별사고 데이터를 사용하여 연령대별 사고 유형 분석하기

가해자연령 변수에 숫자가 아닌 이상한 값이 포함되어 있는지 다시 한번 검토하는 작업을 거치기 위해, unique( ) 함수를 사용하였습니다.

```
df_subset['가해자연령'].unique() # 값이 잘 있는지 확인
```

```
array([ 54.,  57.,  51.,  56.,  33.,  58.,  26.,  30.,  34.,  53.,  27.,
        50.,  66.,  23.,  52.,  44.,  59.,  20.,  40.,  64.,  55.,  41.,
        31.,  49.,  46.,  60.,  45.,  29.,  48.,  47.,  24.,  67.,  37.,
        62.,  71.,  17.,  28.,  25.,  22.,  78.,  38.,  42.,  68.,  43.,
        70.,  61.,  36.,  63.,  69.,  15.,  39.,  65.,  19.,  18.,  35.,
        32.,  21.,  72.,  76.,  73.,  16.,  74.,  75.,  83.,  84.,  80.,
        11.,  86.,  79.,  77.,  12.,  85.,  14.,  13.,  10.,   7.,   8.,
        87.,  91.,  82.,   6.,  94.,   9.,  89.,  81.,   5.,  98.,  92.,
        93.,  88.,  90.,  95.,   4.,  99., 113., 117., 100., 106.,   1.])
```

서울시 개별사고 데이터를 사용하여 연령대별 사고 유형 분석하기

가해자 연령대와 사고 유형을 기준으로 교통사고 건수를 측정하는 작업을 거쳐 df_subset_gp에 저장하고,

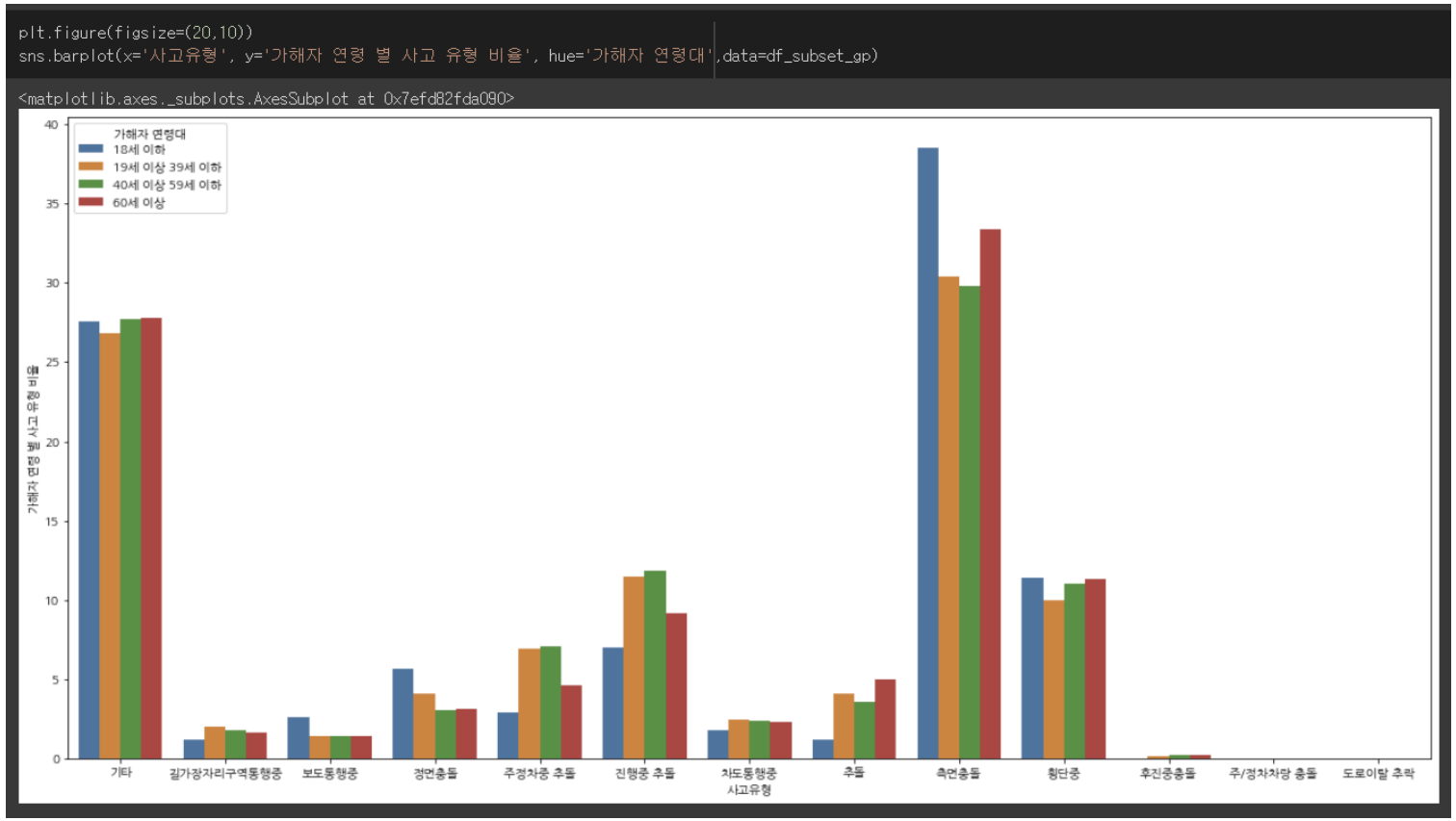
각각 연령대별로 나누어, for문을 사용하여 사고 건수의 총 합계를 구하고, 각 사고 건수를 합계로 나누어주어, 가해자 연령 별 사고 유형 비율이라는 칼럼을 추가해주는 작업을 해주었습니다.

```
df_subset
df_subset_gp = df_subset.groupby(['가해자 연령대','사고유형'])['사고내용'].count().reset_index()
df_subset_gp['가해자 연령대'].unique()[0]
df_subset_gp_items = []
for i in range(len(df_subset_gp['가해자 연령대'].unique())):
  sum = 0
  for j in range(len(df_subset_gp)):
    if df_subset_gp.loc[j, '가해자 연령대'] == df_subset_gp['가해자 연령대'].unique()[i]:
      sum += df_subset_gp.loc[j, '사고내용']
  df_subset_gp_items.append(sum)
df_subset_gp_items

df_subset_gp['가해자 연령 별 사고 유형 비율'] = 0
for i in range(len(df_subset_gp['가해자 연령대'].unique())):
  for j in range(len(df_subset_gp)):
    if df_subset_gp.loc[j, '가해자 연령대'] == df_subset_gp['가해자 연령대'].unique()[i]:
     df_subset_gp.loc[j, '가해자 연령 별 사고 유형 비율'] = df_subset_gp.loc[j, '사고내용'] / df_subset_gp_items[i] * 100
df_subset_gp
```

	가해자 연령대	사고유형	사고내용	가해자 연령 별 사고 유형 비율
0	18세 이하	기타	3126	27.573432
1	18세 이하	길가장자리구역통행중	135	1.190791
2	18세 이하	보도통행중	297	2.619741
3	18세 이하	정면충돌	646	5.698156
4	18세 이하	주정차중 추돌	330	2.910823

서울시 개별사고 데이터를 사용하여 연령대별 사고 유형 분석하기

시각화한 결과, 연령대 별로 각 사고 유형의 비율이 비슷한 결과를 나타내는 것을 볼 수 있었지만,
측면 충돌과 보도통행중, 정면 충돌의 경우 18세 이하 청소년의 비율이 많았으며,
추돌의 경우 19~59세의 성인 및 중년층의 비율이 다른 연령층 보다 많은 비율을 차지하는 것을
확인할 수 있었습니다.



이륜차 및 자전거의 교통사고 데이터 분석하기

이번에는 이륜차 및 자전거가 가해자 차종인 경우의 교통사고의 특징을 분석해보도록 하겠습니다.

```
df_cycle = df[(df['가해자차종'] == '이륜차') |( df['가해자차종'] == '자전거')]
df_cycle.head()
```

	Unnamed: 0	발생일	발생년도	발생월	발생일자	발생시간	요일	발생지_시군구	사고내용	법정동명	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해성별	가해자연령	가해자신체상해정도	피해자차종	피해자성별	피해자연령	피해자신체상해정도
77	77	2010-01-01	2010	1	1	13시	금	구로구	경상	오류동	0	0	1	0	차대차	측면충돌	측면충돌	안전운전 의무 불이행	포장	건조	맑음	교차로	교차로내	이륜차	남	17세	경상	승용차	남	23세	상해없음
109	109	2010-01-01	2010	1	1	22시	금	동대문구	중상	청량리동	0	1	1	0	차대차	기타	기타	신호위반	포장	젖음/습기	흐림	교차로	교차로내	이륜차	기타불명	불명	기타불명	이륜차	남	43세	중상
167	167	2010-01-02	2010	1	2	18시	토	강동구	경상	상일동	0	0	1	0	차대차	정면충돌	정면충돌	교차로 통행방법 위반	포장	젖음/습기	맑음	교차로	교차로내	이륜차	남	48세	경상	이륜차	남	16세	상해없음
168	168	2010-01-02	2010	1	2	18시	토	구로구	경상	오류동	0	0	1	0	차대차	측면충돌	측면충돌	안전운전 의무 불이행	포장	건조	흐림	교차로	교차로부근	이륜차	남	15세	경상	화물차	여	34세	상해없음
186	186	2010-01-03	2010	1	3	00시	일	강북구	경상	수유동	0	0	1	0	차대차	측면충돌	측면충돌	안전거리 미확보	포장	젖음/습기	흐림	단일로	횡단보도상	이륜차	남	18세	경상	승용차	남	47세	상해없음

이륜차 및 자전거의 교통사고 데이터 분석하기

피해자 차종에 따른 분석 결과, 승용차 또는 보행자와 충돌한 경우가 다른 차종보다 압도적으로 많으며,

각각은 이륜차 및 자전거 교통사고는 승용차와 충돌한 경우가 17929건으로 가장 많았으며, 보행자와 충돌한 경우가 9691건으로 두 번째로 많음을 확인할 수 있었습니다.

그리고, 가해자 위반별 비율은 전체적인 경우와, 가해자 차종이 이륜차,자전거인 경우에 어떻게 다른지 분석해보기 위해 df_cyc_gp과 df_gp의 데이터셋으로 나누었습니다.

```
df_cycle.groupby(['피해자차종'])['사고내용'].count().sort_values(ascending=False).reset_index()
```

	피해자차종	사고내용
0	승용차	17929
1	보행자	9691
2	자전거	3566
3	이륜차	3474
4	승합차	2371
5	화물차	2233
6	없음	1963
7	원동기장치자전거	468
8	불명	124
9	건설기계	89
10	특수차	36
11	기타	11
12	개인형이동수단(PM)	6
13	농기계	5
14	사륜오토바이(ATV)	1

```
df_cyc_gp = df_cycle.groupby(['가해자법규위반'])['사고내용'].count().reset_index()

df_cyc_gp['위반별 비율'] = df_cyc_gp['사고내용'] / df_cyc_gp['사고내용'].sum(axis=0) * 100
df_cyc_gp['가해자차종'] = "이륜차|자전거"
```

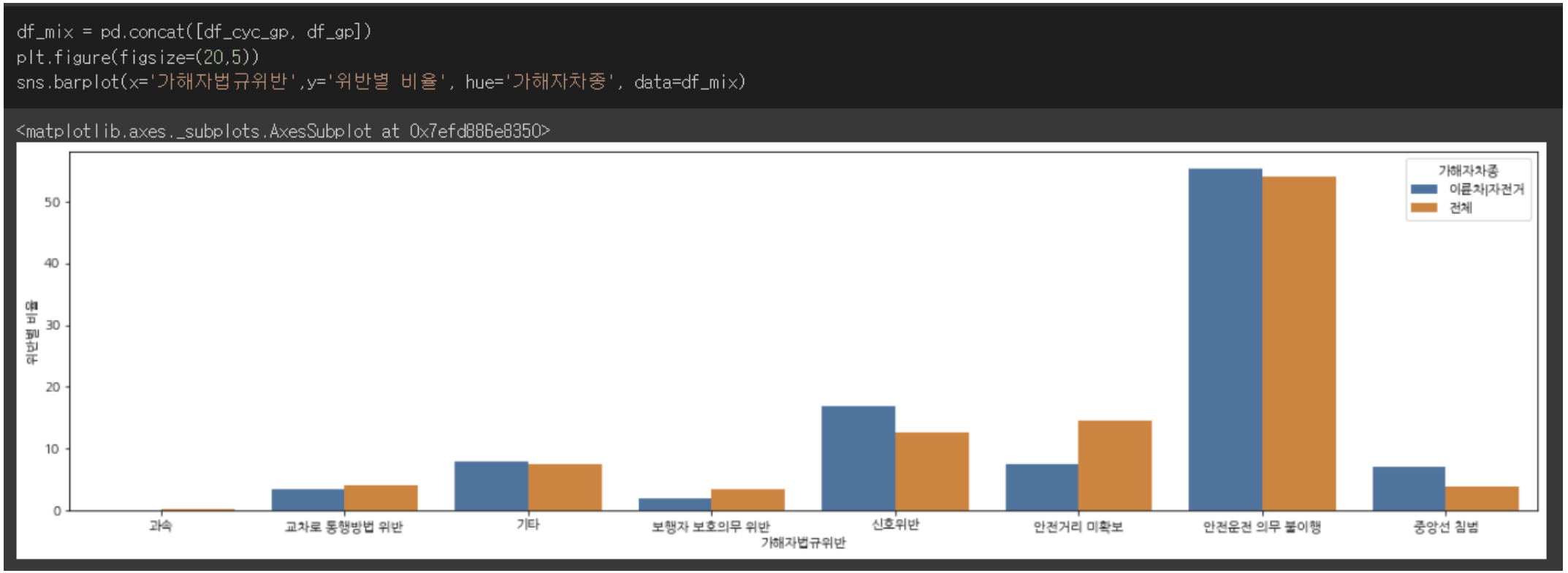
```
df_gp = df_subset.groupby(['가해자법규위반'])['사고내용'].count().reset_index()

df_gp['위반별 비율'] = df_gp['사고내용'] / df_gp['사고내용'].sum(axis=0) * 100
df_gp['가해자차종']="전체"
```

이륜차 및 자전거의 교통사고 데이터 분석하기

각각의 경우를 확인해본 결과, 교차로 통행방법과 안전운전 의무 불이행의 경우는 비슷하였으나,
이륜차 및 자전거가 가해자 차종인 경우
안전거리 미확보와 보행자 보호의무 위반의 경우가 상대적으로 적은 비율을 차지하고,
중앙선 침범 및 신호 위반의 경우에는 상대적으로 많은 비율을 차지하고 있음을 확인할 수 있었습니다.

```
df_mix = pd.concat([df_cyc_gp, df_gp])
plt.figure(figsize=(20,5))
sns.barplot(x='가해자법규위반',y='위반별 비율', hue='가해자차종', data=df_mix)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd886e8350>
```



광주광역시 교통사고 데이터로 sum 모델 검증하기

사용 데이터 : 도로교통공단_광주광역시 개별사고정보(2015~19년)

먼저 데이터프레임의 형태를 살펴보기 위해 head( )로 앞의 3행만 추출해보았습니다.

```
gj_data = pd.read_csv('/content/drive/MyDrive/꿔8로우/예측용데이터/도로교통공단_광주광역시 개별사고정보_15~19.csv', sep=',', encoding='cp949')
gj_data.head()
```

	발생일	발생시간	발생요일	시군구	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자종	가해자성별	가해자연령	가해자신체상해정도	피해자종	피해자성별	피해자연령	피해자신체상해정도
0	2015-01-30	0	금	광산구	경상	0	0	1	0	차량단독	기타	기타	안전운전의무 불이행	포장	서리/결빙	눈	단일로	고가도로위	승용차	남	44세	상해없음	없음	없음	불명	없음
1	2015-03-28	22	토	북구	경상	0	0	2	0	차대차	추돌	진행중 추돌	안전운전의무 불이행	포장	건조	맑음	단일로	기타단일로	승용차	여	28세	상해없음	승용차	남	28세	경상
2	2015-08-03	5	월	북구	경상	0	0	2	0	차대차	추돌	진행중 추돌	안전운전의무 불이행	포장	건조	맑음	단일로	기타단일로	승용차	남	46세	경상	승합차	남	54세	경상
3	2015-09-07	20	월	광산구	경상	0	0	1	0	차대차	추돌	주정차 중 추돌	안전운전의무 불이행	포장	건조	맑음	단일로	기타단일로	승용차	여	54세	경상	화물차	남	33세	상해없음
4	2015-10-03	13	토	광산구	경상	0	0	2	0	차대차	추돌	진행중 추돌	안전거리 미확보	포장	건조	맑음	단일로	터널안	승용차	남	52세	상해없음	승용차	남	35세	경상

광주광역시 교통사고 데이터로 sum 모델 검증하기

다른 지역의 칼럼명과 맞춰주기 위한 작업과, 새로운 열(발생일_시간, 사고건수, 발생일, 부상자수 등)을 추가하기 위한 전처리 작업을 거쳐서

다음과 같은 데이터프레임으로 만들어주도록 합니다.

```
gj_data.rename(columns={'시군구': '발생지_시군구', '발생요일':'요일','가해자성별': '가해성별', }, inplace=True)
```

```
# 발생시간을 00:00꼴의 형태로 변환하기. (다른 지역과 맞춰주기 위해..)
# (발생일_시간, 발생일_시간_시군구 칼럼에 추가하기 위한 작업)
gj_data['발생시간'] = [f"{i:#02d}:00" for i in list(gj_data['발생시간'])]
```

```
gj_data['발생일']=[str(i) for i in gj_data['발생일']]
gj_data['발생일_시간'] = gj_data['발생일']+" "+ gj_data['발생시간'].astype(str)
gj_data['발생일_시간_시군구'] = gj_data['발생일']+" "+gj_data['발생시간'].astype(str)+" "+gj_data['발생지_시군구']
gj_data['사고건수'] = [1 for i in gj_data['발생일']]
gj_data['발생일'] = pd.to_datetime(gj_data['발생일'], format='%Y-%m-%d')
gj_data['발생년도'] = gj_data['발생일'].dt.year
gj_data['발생월'] = gj_data['발생일'].dt.month
gj_data['발생일자'] = gj_data['발생일'].dt.day
gj_data['부상자수'] = gj_data['사망자수']+gj_data['중상자수']+gj_data['경상자
```

gj_data.head()

	발생일	발생시간	요일	발생지_시군구	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해자성별	가해자연령	가해자신체상해정도	피해자차종	피해자성별	피해자연령	피해자신체상해정도	발생일_시간	발생일_시간_시군구	사고건수	발생년도	발생월	발생일자	부상자수
0	2015-01-30	00:00	금	광산구	경상	0	0	1	0	차량단독	기타	기타	안전운전의무불이행	포장	서리/결빙	눈	단일로	고가도로위	승용차	남	44세	상해없음	없음	없음	불명	없음	2015-01-30 00:00	2015-01-30 00:00 광산구	1	2015	1	30	1
1	2015-03-28	22:00	토	북구	경상	0	0	2	0	차대차	추돌	진행중추돌	안전운전의무불이행	포장	건조	맑음	단일로	기타단일로	승용차	여	28세	상해없음	승용차	남	28세	경상	2015-03-28 22:00	2015-03-28 22:00 북구	1	2015	3	28	2
2	2015-08-03	05:00	월	북구	경상	0	0	2	0	차대차	추돌	진행중추돌	안전운전의무불이행	포장	건조	맑음	단일로	기타단일로	승용차	남	46세	경상	승합차	남	54세	경상	2015-08-03 05:00	2015-08-03 05:00 북구	1	2015	8	3	2
3	2015-09-07	20:00	월	광산구	경상	0	0	1	0	차대차	추돌	주정차중추돌	안전운전의무불이행	포장	건조	맑음	단일로	기타단일로	승용차	여	54세	경상	화물차	남	33세	상해없음	2015-09-07 20:00	2015-09-07 20:00 광산구	1	2015	9	7	1
4	2015-10-03	13:00	토	광산구	경상	0	0	2	0	차대차	추돌	진행중추돌	안전거리미확보	포장	건조	맑음	단일로	터널안	승용차	남	52세	상해없음	승용차	남	35세	경상	2015-10-03 13:00	2015-10-03 13:00 광산구	1	2015	10	3	2

광주광역시 교통사고 데이터로 sum 모델 검증하기

발생일_시간, 요일, 발생년도, 발생월, 발생일자로 묶어
각각 부상자수, 사망자수, 중상자수, 경상자수,부상신고자수를 구하는 작업을 거칩니다.

```
new_data1 = gj_data.groupby(['발생일_시간', '요일', '발생년도','발생월','발생일자']).sum()[['사망자수', '중상자수', '부상신고자수', '사고건수', '부상자수', '경상자수']]
new_data1 = new_data1.reset_index(drop=False)
new_data1 = new_data1.reindex(columns = ['발생일_시간','사고건수', '발생년도', '발생월', '발생일자', '요일','부상자수','사망자수', '중상자수', '경상자수','부상신고자수'])
new_data1
```

	발생일_시간	사고건수	발생년도	발생월	발생일자	요일	부상자수	사망자수	중상자수	경상자수	부상신고자수
0	2015-01-01 00:00	3	2015	1	1	목	4	0	0	4	0
1	2015-01-01 01:00	1	2015	1	1	목	1	0	0	1	0
2	2015-01-01 02:00	3	2015	1	1	목	4	0	0	3	1
3	2015-01-01 03:00	5	2015	1	1	목	7	0	1	6	0
4	2015-01-01 04:00	4	2015	1	1	목	8	0	0	8	0
...	...	...	...	...	...	...	...	...	...	...	...
23926	2019-12-31 18:00	2	2019	12	31	화	2	0	0	2	0
23927	2019-12-31 19:00	1	2019	12	31	화	2	0	0	2	0
23928	2019-12-31 20:00	2	2019	12	31	화	3	0	0	3	0
23929	2019-12-31 21:00	1	2019	12	31	화	1	0	0	1	0
23930	2019-12-31 22:00	3	2019	12	31	화	4	0	0	3	1

23931 rows × 11 columns

광주광역시 교통사고 데이터로 sum 모델 검증하기

연령 데이터와 발생시간을 숫자로 변환시키는 작업을 거친 후,
모든 열의 데이터를 정수형 데이터로 변환하는 작업을 하기 위해 mapping 함수를 적용하였습니다.

```python
#연령 데이터 전처리 (불명은 0세 처리, 정수형으로 변환하기 위해서 뒤에 '세'라는 글자 빼는 작업을 하겠습니다.)
def age_to_num(df):
    for c, i in enumerate(df):
        if i in ['불명','없음']:
            df[c] = '0세'
    df = [int(i[:-1]) for i in df]
    return df

# 가해자, 피해자 연령에 각각 함수를 적용시키겠습니다.

gj_data['가해자연령'] = age_to_num(gj_data['가해자연령'])
gj_data['피해자연령'] = age_to_num(gj_data['피해자연령'])
```

```python
#시간 데이터 전처리 (뒤에 :00이라는 글자를 빼고, 정수형으로 변환시키는 작업을 해주겠습니다.)
gj_data['발생시간'] = [int(i[:-3]) for i in gj_data['발생시간']]
```

```python
new_data2 = gj_data.copy()

# 정수형 데이터로 변환하는 작업을 해드리겠습니다~!~!~!~

for i in ['요일', '발생지_시군구', '사고내용', '사고유형_대분류', '사고유형_중분류', '사고유형', '가해자법규위반', '노면상태_대분류', '노면상태', '기상상태', '도로형태_대분류', '도로형태', '가해자차종', '가해성별', '가해지
  mapping = {label: idx for idx, label in enumerate(np.unique(gj_data[i]))}
  print(mapping)
  new_data2[i] = gj_data[i].map(mapping)
```

```
{'금': 0, '목': 1, '수': 2, '월': 3, '일': 4, '토': 5, '화': 6}
{'광산구': 0, '남구': 1, '동구': 2, '북구': 3, '서구': 4}
{'경상': 0, '부상신고': 1, '사망': 2, '중상': 3}
{'차대사람': 0, '차대차': 1, '차량단독': 2}
{'공작물충돌': 0, '기타': 1, '길가장자리구역통행중': 2, '도로이탈': 3, '보도통행중': 4, '전도': 5, '전도전복': 6, '전복': 7, '정면충돌': 8, '주/정차차량 충돌': 9, '차도통행중': 10, '추돌': 11, '측면충돌': 12, '횡단중': 13,
{'공작물충돌': 0, '기타': 1, '길가장자리구역통행중': 2, '도로이탈 기타': 3, '도로이탈 추락': 4, '보도통행중': 5, '전도': 6, '전도전복': 7, '전복': 8, '정면충돌': 9, '주/정차차량 충돌': 10, '주정차중 추돌': 11, '진행중 추돌'
{'과속': 0, '교차로 통행방법 위반': 1, '기타': 2, '보행자 보호의무 위반': 3, '신호위반': 4, '안전거리 미확보': 5, '안전운전 의무 불이행': 6, '중앙선 침범': 7}
{'비포장': 0, '포장': 1}
{'건조': 0, '기타': 1, '서리/결빙': 2, '적설': 3, '젖음/습기': 4, '침수': 5, '해빙': 6}
{'기타/불명': 0, '눈': 1, '맑음': 2, '비': 3, '안개': 4, '흐림': 5}
{'교차로': 0, '기타': 1, '기타/불명': 2, '단일로': 3, '불명': 4}
{'고가도로위': 0, '교량위': 1, '교차로내': 2, '교차로부근': 3, '교차로횡단보도내': 4, '기타': 5, '기타/불명': 6, '기타단일로': 7, '불명': 8, '지하차도(도로)내': 9, '터널안': 10, '횡단보도부근': 11, '횡단보도상': 12}
{'개인형이동수단(PM)': 0, '건설기계': 1, '기타': 2, '농기계': 3, '불명': 4, '사륜오토바이(ATV)': 5, '승용차': 6, '승합차': 7, '원동기장치자전거': 8, '이륜차': 9, '자전거': 10, '특수차': 11, '화물차': 12}
{'기타불명': 0, '남': 1, '여': 2}
{'경상': 0, '기타불명': 1, '부상신고': 2, '사망': 3, '상해없음': 4, '중상': 5}
{'개인형이동수단(PM)': 0, '건설기계': 1, '기타': 2, '농기계': 3, '보행자': 4, '불명': 5, '사륜오토바이(ATV)': 6, '승용차': 7, '승합차': 8, '없음': 9, '원동기장치자전거': 10, '이륜차': 11, '자전거': 12, '특수차': 13, '화물차
{'기타불명': 0, '남': 1, '없음': 2, '여': 3}
{'경상': 0, '기타불명': 1, '부상신고': 2, '사망': 3, '상해없음': 4, '없음': 5, '중상': 6}
```

광주광역시 교통사고 데이터로 sum 모델 검증하기

다음과 같이 변환된 것을 알 수 있으며, 발생지_시군구와 발생년도로 그룹화하여 mean_data를 만들어줍니다.

new_data2.head()

	발생일	발생시간	요일	발생지_시군구	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해성별	가해자연령	가해자신체상해정도	피해자차종	피해자성별	피해자연령	피해자신체상해정도	발생일_시간	발생일_시간_시군구	사고건수	발생년도	발생월	발생일자	부상자수
0	2015-01-30	0	0	0	0	0	0	1	0	2	1	1	6	1	2	1	3	0	6	1	44	4	9	2	0	5	2015-01-30 00:00	2015-01-30 00:00 광산구	1	2015	1	30	1
1	2015-03-28	22	5	3	0	0	0	2	0	1	11	12	6	1	0	2	3	7	6	2	28	4	7	1	28	0	2015-03-28 22:00	2015-03-28 22:00 북구	1	2015	3	28	2
2	2015-08-03	5	3	3	0	0	0	2	0	1	11	12	6	1	0	2	3	7	6	1	46	0	8	1	54	0	2015-08-03 05:00	2015-08-03 05:00 북구	1	2015	8	3	2
3	2015-09-07	20	3	0	0	0	0	1	0	1	11	11	6	1	0	2	3	7	6	2	54	0	14	1	33	4	2015-09-07 20:00	2015-09-07 20:00 광산구	1	2015	9	7	1
4	2015-10-03	13	5	0	0	0	0	2	0	1	11	12	5	1	0	2	3	10	6	1	52	4	7	1	35	0	2015-10-03 13:00	2015-10-03 13:00 광산구	1	2015	10	3	2

```
mean_data = new_data2.groupby(['발생지_시군구','발생년도']).mean()
mean_data['사고건수'] = new_data2.groupby(['발생지_시군구','발생년도'])['사고건수'].sum() #사고건수만 sum
mean_data = mean_data.reset_index(drop=False)
mean_data.head()
```

	발생지_시군구	발생년도	발생시간	요일	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해성별	가해자연령	가해자신체상해정도
0	0	2015	13.829500	2.868500	0.578500	0.013500	0.194500	1.394000	0.050000	0.879500	9.773000	11.444500	4.848000	0.997500	0.645500	2.290000	1.407500	4.589000	7.033000	1.171000	42.528500	3.177000
1	0	2016	13.953548	2.883104	0.651863	0.015824	0.229709	1.359877	0.055641	0.866258	10.210311	12.189893	4.821337	0.998979	0.590607	2.198571	1.551812	4.857580	6.905564	1.168453	42.333333	3.173047
2	0	2017	13.682167	2.839550	0.654062	0.017374	0.228411	1.311702	0.037302	0.855391	9.794073	12.161983	4.859479	0.998978	0.361778	2.145631	1.308125	4.436893	6.943281	1.179356	42.449668	3.285130
3	0	2018	13.960204	2.838265	0.633163	0.013265	0.223980	1.376531	0.041837	0.883673	9.277551	11.551020	4.881122	0.998980	0.430102	2.132143	1.339796	4.484184	6.959184	1.181633	44.358163	3.227041
4	0	2019	13.910756	2.837482	0.551902	0.007046	0.184124	1.396430	0.077971	0.891029	8.947863	11.121653	4.779239	0.999061	0.409112	2.155002	1.371066	4.506811	6.952090	1.209018	44.004697	3.141381

광주광역시 교통사고 데이터로 sum 모델 검증하기

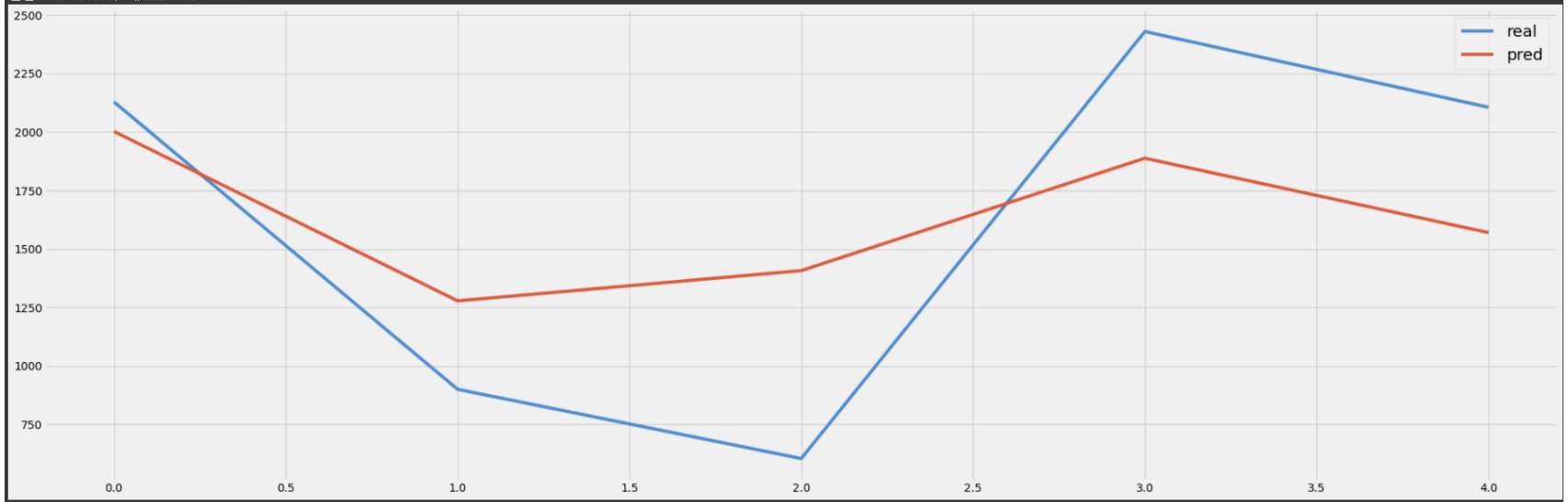
Feature_names에 다음 항목을 적용하여, 랜덤 포레스트 모델을 적용시켜보았습니다.

Mean을 사용한 경우에는 예측 정확도가 많이 떨어지는 것을 확인할 수 있네요..

```
feature_names = ['발생시간','요일', '발생년도', '발생월', '발생일자', '노면상태_대분류', '노면상태', '기상상태', '도로형태', '도로형태_대분류']
```

```
X_train, X_test, y_train, y_test = data_split(mean_data, feature_names)
forest_fit()
```

```
훈련 MSE: 52732.250, 테스트 MSE: 277092.025
훈련 R^2: 0.886, 테스트 R^2: 0.487
```




```
CV 정확도 점수: [-3.91313171e+01 -2.51943896e+04 -1.45316792e+02 -3.64582632e+03
 -3.42132783e+01 -9.98712946e+02 -1.25543153e+05 -3.47744216e+03
 -1.28276522e+01 -3.33645387e+02]
CV 정확도: -15942.466 +/- 37256.859
```

광주광역시 교통사고 데이터로 sum 모델 검증하기

이번에는 발생지_시군구와 발생년도를 기준으로 그룹화하여 각각의 데이터를 합한 데이터셋을 만들어 보겠습니다.

```
sum_data = new_data2.groupby(['발생지_시군구','발생년도']).sum()
sum_data = sum_data.reset_index(drop=False)
sum_data.head()
```

	발생지_시군구	발생년도	발생시간	요일	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해성별	가해자연령	가해자신체상해정도	피해자차종	피해자성별	피해자연령	피해자신체상해정도	사고건수	발생월	발생일자	부상자수
0	0	2015	27659	5737	1157	27	389	2788	100	1759	19546	22889	9696	1995	1291	4580	2815	9178	14066	2342	85057	6354	15316	3144	78546	2759	2000	13232	31351	3304
1	0	2016	27335	5648	1277	31	450	2664	109	1697	20002	23880	9445	1957	1157	4307	3040	9516	13528	2289	82931	6216	14892	3065	76715	2931	1959	13029	31153	3254
2	0	2017	26776	5557	1280	34	447	2567	73	1674	19167	23801	9510	1955	708	4199	2560	8683	13588	2308	83074	6429	14904	3076	78332	2826	1957	12961	31657	3121
3	0	2018	27362	5563	1241	26	439	2698	82	1732	18184	22640	9567	1958	843	4179	2626	8789	13640	2316	86942	6325	14951	3141	80304	2736	1960	13262	31054	3245
4	0	2019	29616	6041	1175	15	392	2973	166	1897	19050	23678	10175	2127	871	4588	2919	9595	14801	2574	93686	6688	16099	3371	86447	2757	2129	14450	33528	3546

광주광역시 교통사고 데이터로 sum 모델 검증하기

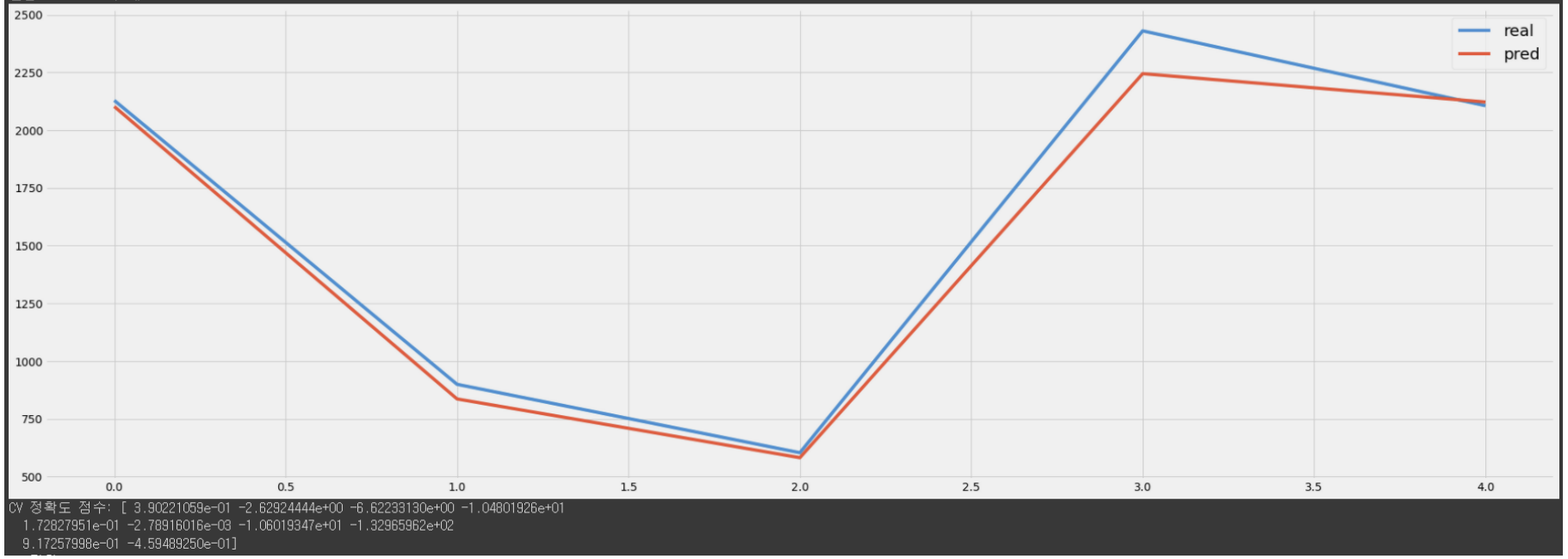
Feature_names에 다음 항목을 적용하여, 랜덤 포레스트 모델을 적용시켜보았습니다.

Sum()을 사용한 경우에는, mean()을 사용한 경우보다 예측을 훨씬 정확하게 하는 것을 확인할 수 있습니다.

```
feature_names = ['발생시간','요일', '발생년도', '발생월', '발생일자', '노면상태_대분류', '노면상태', '기상상태', '도로형태', '도로형태_대분류']
```

```
X_train, X_test, y_train, y_test = data_split(sum_data, feature_names)
forest_fit()
```

```
훈련 MSE: 368.311, 테스트 MSE: 7953.882
훈련 R^2: 0.999, 테스트 R^2: 0.985
```

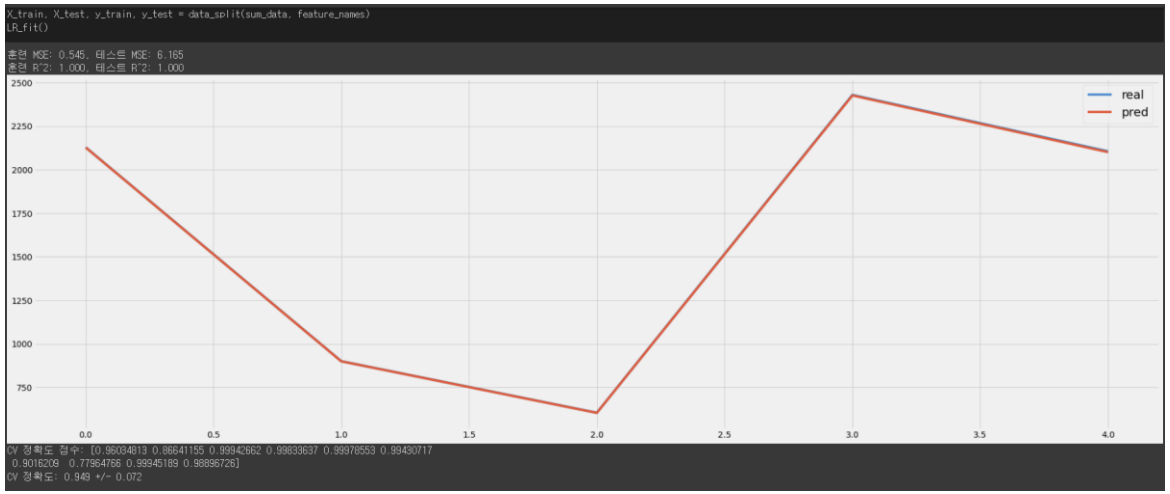


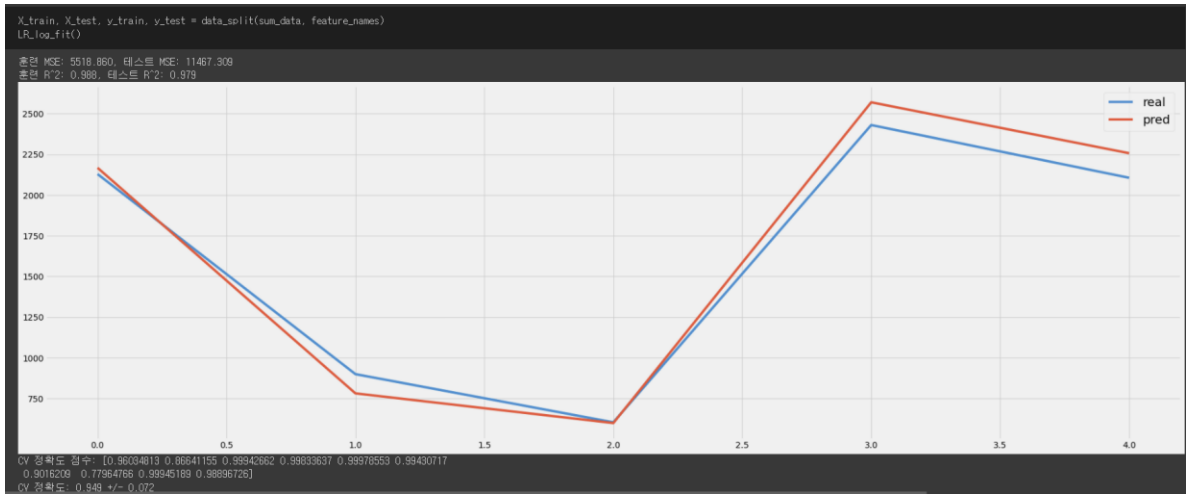

```
CV 정확도 점수: [ 3.90221059e-01 -2.62924444e+00 -6.62233130e+00 -1.04801926e+01
  1.72827951e-01 -2.78916016e-03 -1.06019347e+01 -1.32965962e+02
  9.17257998e-01 -4.59489250e-01]
```

광주광역시 교통사고 데이터로 sum 모델 검증하기

선형 회귀 모델을 적용한 경우에는 거의 예측 정확도가 1에 가까웠으며,
선형 로그 회귀 모델을 적용한 경우에도 훨씬 높은 예측 정확도를 보인 것을 확인할 수 있었습니다.

```
feature_names = ['발생시간','요일', '발생년도', '발생월', '발생일자', '노면상태_대분류', '노면상태', '기상상태', '도로형태', '도로형태_대분류']
```





광주광역시 교통사고 데이터로 sum 모델 검증하기

이번에는 요일, 발생년도, 발생월, 발생일자, 발생시간, 발생지_시군구를 기준으로 묶어 데이터를 sum()하여 사고건수를 예측해보도록 하겠습니다.

```
col = ['사고내용', '사망자수', '중상자수', '경상자수','부상신고자수', '사고유형_대분류', '사고유형_중분류', '사고유형', '가해자법규위반', '노면상태_대분류', '노면상태','기상상태',
       '도로형태_대분류', '도로형태', '가해자차종', '가해성별', '가해자연령', '가해자신체상해정도', '피해자차종', '피해자성별', '피해자연령', '피해자신체상해정도',  '사고건수','부상자수']
group_data = new_data2.groupby(['요일', '발생년도','발생월','발생일자','발생시간','발생지_시군구']).sum()[col]
group_data = group_data.reset_index(drop=False)
group_data.head()
```

	요일	발생년도	발생월	발생일자	발생시간	발생지_시군구	사고내용	사망자수	중상자수	경상자수	부상신고자수	사고유형_대분류	사고유형_중분류	사고유형	가해자법규위반	노면상태_대분류	노면상태	기상상태	도로형태_대분류	도로형태	가해자차종	가해성별	가해자연령	가해자신체상해정도	피해자차종	피해자성별	피해자연령	피해자신체상해정도	사고건수	부상자수
0	0	2015	1	2	1	3	0	0	0	1	0	2	0	0	6	1	3	1	3	7	6	1	55	4	9	2	0	5	1	1
1	0	2015	1	2	8	0	0	0	0	4	0	2	19	21	12	2	5	2	6	8	13	2	92	8	14	2	74	0	2	4
2	0	2015	1	2	9	0	0	0	0	1	0	2	0	0	6	1	3	1	0	3	6	1	42	4	9	2	0	5	1	1
3	0	2015	1	2	9	2	3	0	1	0	0	2	1	1	2	1	2	2	3	7	7	1	52	4	9	2	0	5	1	1
4	0	2015	1	2	10	4	3	0	1	0	0	0	10	13	6	1	2	1	3	7	6	1	34	4	4	1	68	6	1	1

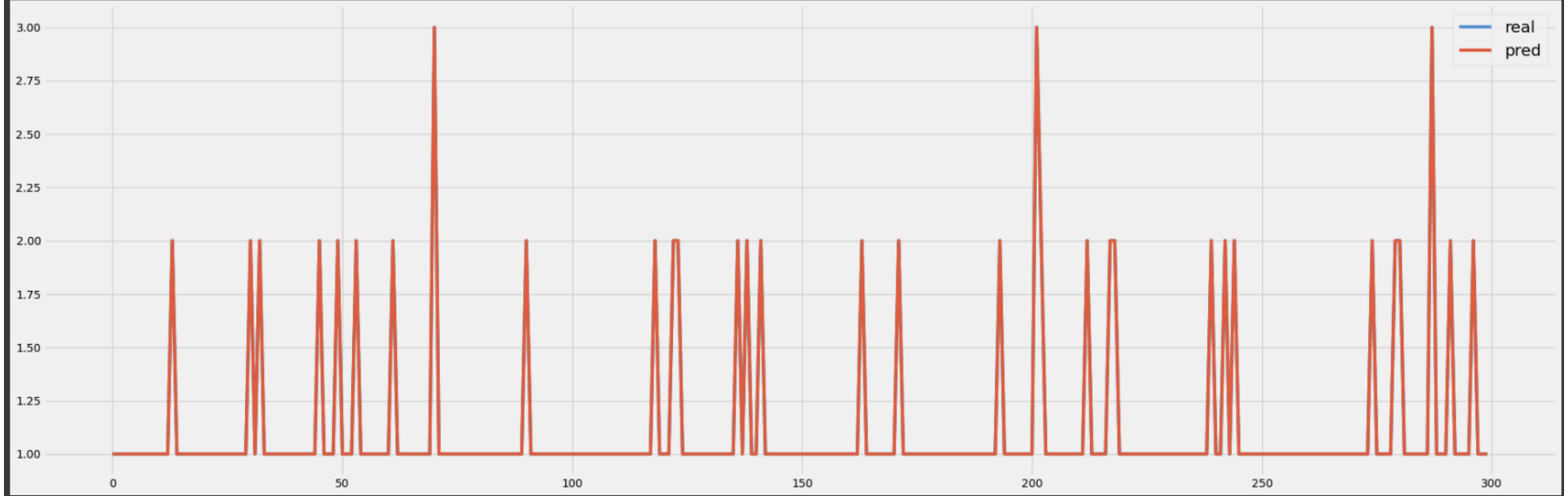
광주광역시 교통사고 데이터로 sum 모델 검증하기

Feature_names에 다음 항목을 적용하여, 랜덤 포레스트 모델을 적용시켜본 결과,
거의 1에 가까운 예측 정확도를 보인 것을 확인할 수 있었습니다.

```
feature_names = ['발생시간','요일', '발생년도', '발생월', '발생일자', '노면상태_대분류', '노면상태', '기상상태', '도로형태', '도로형태_대분류']
```

```
X_train, X_test, y_train, y_test = data_split(group_data, feature_names)
forest_fit()
```

```
훈련 MSE: 0.000, 테스트 MSE: 0.000
훈련 R^2: 1.000, 테스트 R^2: 1.000
```




```
CV 정확도 점수: [0.99998715 0.99771153 0.9799087  0.99969922 0.99559596 1.
 0.99909508 0.99993513 0.99997571 0.99998954]
CV 정확도: 0.997 +/- 0.006
```

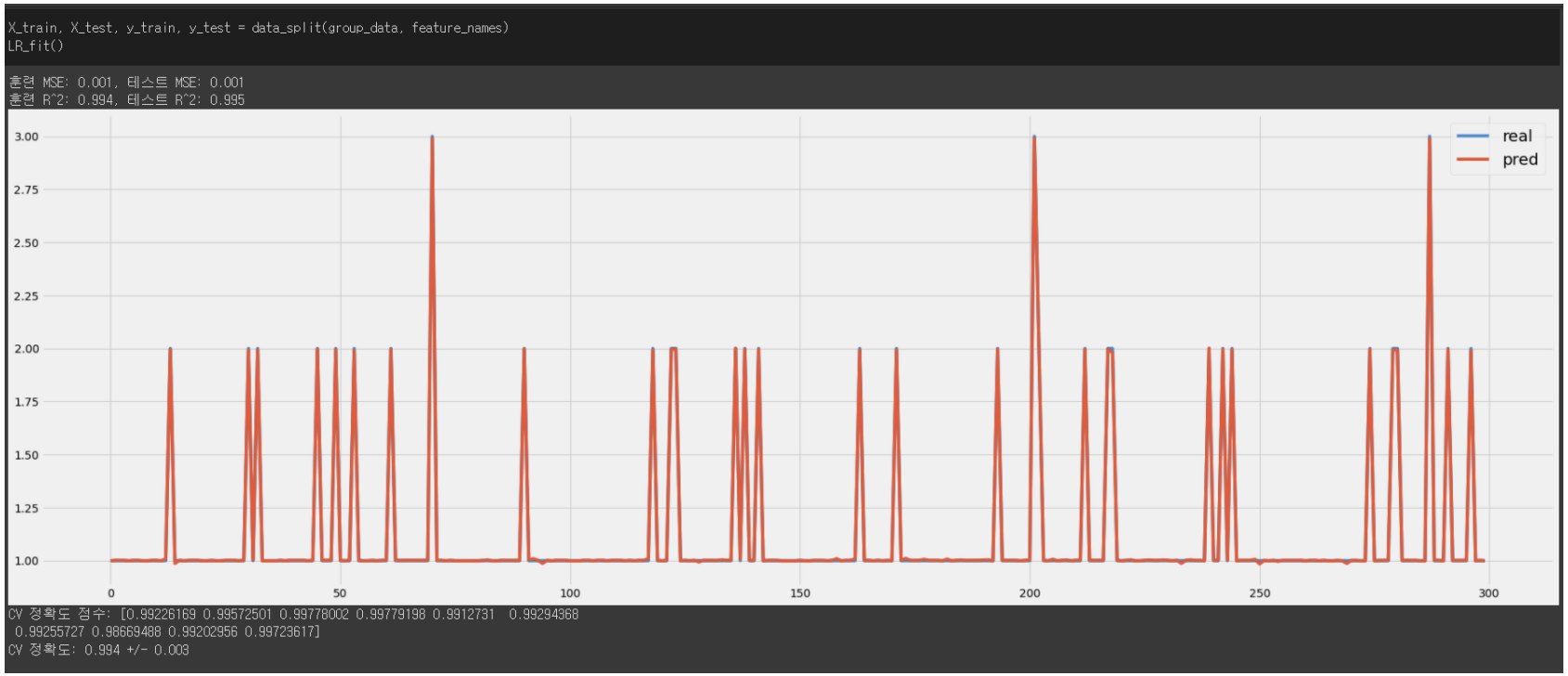
광주광역시 교통사고 데이터로 sum 모델 검증하기

Feature_names에 다음 항목을 적용하여, 선형 회귀 모델을 적용한 경우에도,
거의 1에 가까운 예측 정확도를 보인 것을 확인할 수 있었습니다.

```
feature_names = ['발생시간','요일', '발생년도', '발생월', '발생일자', '노면상태_대분류', '노면상태', '기상상태', '도로형태', '도로형태_대분류']
```

```
X_train, X_test, y_train, y_test = data_split(group_data, feature_names)
LR_fit()
```

```
훈련 MSE: 0.001, 테스트 MSE: 0.001
훈련 R^2: 0.994, 테스트 R^2: 0.995
```




```
CV 정확도 점수: [0.99226169 0.99572501 0.99778002 0.99779198 0.9912731  0.99294368
 0.99255727 0.98669488 0.99202956 0.99723617]
CV 정확도: 0.994 +/- 0.003
```

광주광역시 교통사고 데이터로 sum 모델 검증하기

그룹화한 데이터로 랜덤 포레스트 모델을 돌린 결과,

기상상태, 도로형태, 노면상태 이 세 변수가 포함되었을 때 높은 R^2값과 낮은 MSE값을 보인 것을 확인할 수 있었으며, 교통사고 사고 건수에 높은 영향을 미치는 변수로 파악하였습니다.

```
forest_top5(group_data, feature_names ,2)
```

```
feature_names2 =  ['기상상태', '도로형태', '도로형태_대분류','노면상태' ]
forest_top5(group_data, feature_names2 ,3)
```

```
 랜덤포레스트 변수:  ['기상상태', '도로형태', '도로형태_대분류']
훈련 MSE: 0.003, 테스트 MSE: 0.006
훈련 R^2: 0.982, 테스트 R^2: 0.964

 랜덤포레스트 변수:  ['기상상태', '도로형태', '노면상태']
훈련 MSE: 0.002, 테스트 MSE: 0.003
훈련 R^2: 0.989, 테스트 R^2: 0.982

 랜덤포레스트 변수:  ['기상상태', '도로형태_대분류', '노면상태']
훈련 MSE: 0.006, 테스트 MSE: 0.006
훈련 R^2: 0.962, 테스트 R^2: 0.966

 랜덤포레스트 변수:  ['도로형태', '도로형태_대분류', '노면상태']
훈련 MSE: 0.018, 테스트 MSE: 0.030
훈련 R^2: 0.880, 테스트 R^2: 0.820
```

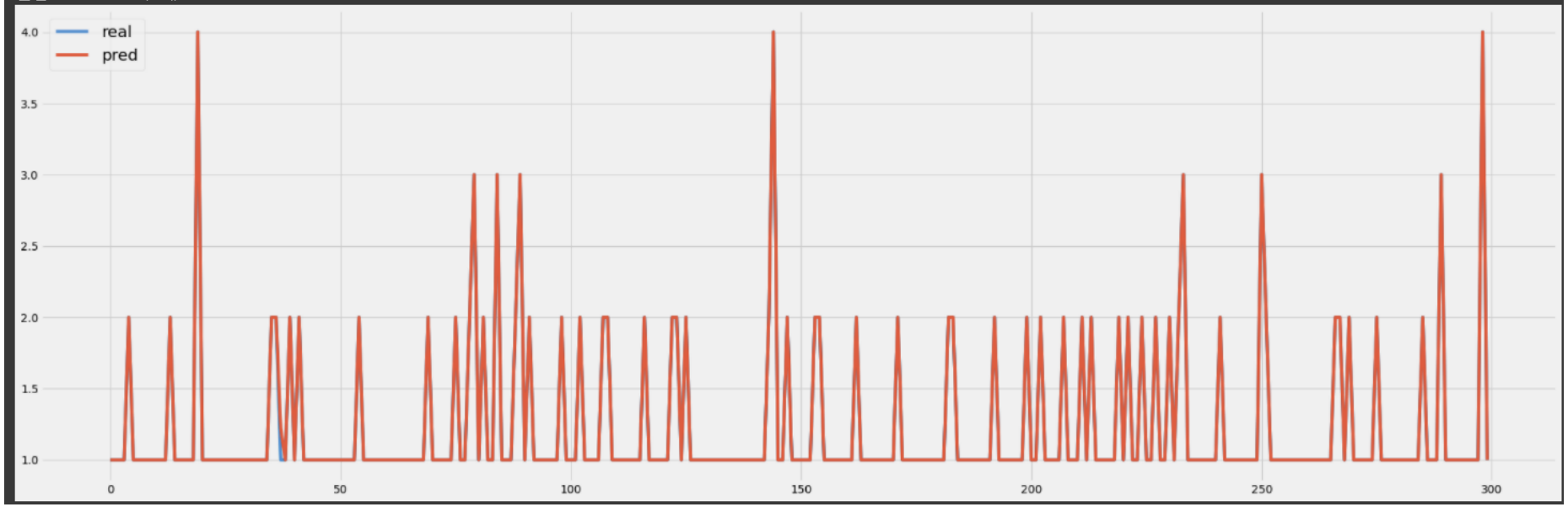
제주도 교통사고 데이터로 sum 모델 검증하기

Feature_names에 다음 항목을 적용하여, 랜덤 포레스트 모델을 적용시켜본 결과,
거의 1에 가까운 예측 정확도를 보인 것을 확인할 수 있었습니다.

```
feature_names = ['발생시간','요일', '발생년도', '발생월', '발생일자', '노면상태_대분류', '노면상태', '기상상태', '도로형태', '도로형태_대분류']
```

```
X_train, X_test, y_train, y_test = data_split(group_data, feature_names)
forest_fit()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py:392: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squ
  FutureWarning,
훈련 MSE: 0.000, 테스트 MSE: 0.000
훈련 R^2: 0.999, 테스트 R^2: 1.000
```



제주광역시 교통사고 데이터로 sum 모델 검증하기

그룹화한 데이터로 랜덤 포레스트 모델을 돌린 결과,

기상상태, 도로형태, 노면상태 이 세 변수가 포함되었을 때 높은 R^2값과 낮은 MSE값을 보인 것을 확인할 수 있었으며, 교통사고 사고 건수에 높은 영향을 미치는 변수로 파악하였습니다.

```
forest_top5(group_data, feature_names ,2)
```

```
feature_names2 =  ['기상상태', '도로형태', '도로형태_대분류','노면상태' ]
forest_top5(group_data, feature_names2 ,3)
```

```
 랜덤포레스트 변수:  ['기상상태', '도로형태', '도로형태_대분류']
훈련 MSE: 0.002, 테스트 MSE: 0.007
훈련 R^2: 0.991, 테스트 R^2: 0.972

 랜덤포레스트 변수:  ['기상상태', '도로형태', '노면상태']
훈련 MSE: 0.004, 테스트 MSE: 0.008
훈련 R^2: 0.979, 테스트 R^2: 0.968

 랜덤포레스트 변수:  ['기상상태', '도로형태_대분류', '노면상태']
훈련 MSE: 0.009, 테스트 MSE: 0.013
훈련 R^2: 0.954, 테스트 R^2: 0.946

 랜덤포레스트 변수:  ['도로형태', '도로형태_대분류', '노면상태']
훈련 MSE: 0.010, 테스트 MSE: 0.017
훈련 R^2: 0.949, 테스트 R^2: 0.930
```

재라벨링 및 타 지역 교통사고 예측 모델 검증 작업

서울시 교통사고 데이터(2010~18년), 경기도 데이터(2016~18년)는 2018년 데이터를 test 데이터로,
광주시 교통사고 데이터(2015~19년)은 2019년 데이터를 test 데이터로,
그리고 세종시, 제주도 데이터는 2017년의 1개년치 데이터밖에 없으므로, 1~25일 데이터를 train,
26~말일 데이터를 test 데이터로 분할하는 함수를 만들어줍니다.

```
def data_split(data, feature_names):

  X_train = data[(data['발생년도']<2018)][feature_names]
  X_test = data[(data['발생년도']==2018)][feature_names]
  y_train = data[(data['발생년도']<2018)]['사고건수']
  y_test = data[(data['발생년도']==2018)]['사고건수']
  y_test= y_test.reset_index(drop=True)

  return X_train, X_test, y_train, y_test
```

```
def data_split_gj(data, feature_names):

  X_train = data[(data['발생년도']<2019)][feature_names]
  X_test = data[(data['발생년도']==2019)][feature_names]
  y_train = data[(data['발생년도']<2019)]['사고건수']
  y_test = data[(data['발생년도']==2019)]['사고건수']
  y_test= y_test.reset_index(drop=True)

  return X_train, X_test, y_train, y_test
```

```
def data_split_sjjj(data, feature_names):

  X_train = data[(data['발생일자']<=25)][feature_names]
  X_test = data[(data['발생일자']>25)][feature_names]
  y_train = data[(data['발생일자']<=25)]['사고건수']
  y_test = data[(data['발생일자']>25)]['사고건수']
  y_test= y_test.reset_index(drop=True)

  return X_train, X_test, y_train, y_test
```

feature_names 리스트에 요일, 발생년도, 월, 일, 노면상태, 기상상태,
도로형태 변수를 넣어, 특성 데이터(X_train, X_test)에 넣어주도록 하고,
'사고건수'에 해당하는 데이터를 레이블 데이터(y_train,y_test)에 넣습니다.

재라벨링 및 타 지역 교통사고 예측 모델 검증 작업

```
def data_split1(data1, data2, feature_names):

  X_train = data1[feature_names]
  X_test = data2[feature_names]
  y_train = data1['사고건수표준화']
  y_test = data2['사고건수표준화']
  y_test= y_test.reset_index(drop=True)

  return X_train, X_test, y_train, y_test
```

```
def data_split2(data1, data2, feature_names):

  X_train = data1[feature_names]
  X_test = data2[feature_names]
  y_train = data1['사고건수최대최소화']
  y_test = data2['사고건수최대최소화']
  y_test= y_test.reset_index(drop=True)

  return X_train, X_test, y_train, y_test
```

서울시 교통사고 예측 모델을 타 지역에도 일반화하기 위한 작업으로,

data1에는 발생년도, 월, 일, 도로형태로 그룹화하고, 요일, 노면상태, 기상상태 변수에 mean( )함수를 적용하여,

각각의 그룹에 해당하는 사고건수를 담고 있는 서울시 교통사고 데이터셋을 넣어주도록 하고,

data2에는 각각의 지역에 해당하는 그룹화된 교통사고 데이터셋을 넣어주도록 합니다.

data_split1과 data_split2 함수를 만들어주는 이유는 각 지역의 생활인구 수가 다르기 때문에, 지역 간의 교통사고 발생 건수 스케일을 조정하기 위해서입니다.

data_split1 함수는 사고건수를 표준화 시켜주는 함수로,

(사고건수 – 사고건수의 평균) / (사고건수의 표준편차) 식을 적용,

data_split2 함수는 사고건수를 최대-최소 정규화 시켜주는 함수로,

(사고건수 – 사고건수의 최솟값) / (사고건수의 최댓값 – 사고건수의 최솟값)의 식을 적용합니다.

재라벨링 및 타 지역 교통사고 예측 모델 검증 작업

```
for i in ['발생시간', '요일',  '노면상태_대분류', '노면상태', '기상상태', '도로형태_대분류', '도로형태']:
  mapping = {label: idx for idx, label in enumerate(main_data.groupby(i).sum()['사고건수'].sort_values().index)}
  print(mapping)
  new_data_seoul[i] = main_data[i].map(mapping)
```

```
{'4': 0, '3': 1, '5': 2, '6': 3, '2': 4, '7': 5, '1': 6, '10': 7, '11': 8, '12': 9, '9': 10, '13': 11, '0': 12, '14': 13, '8': 14, '15': 15, '21': 16, '16': 17, '23': 18, '20': 19, '17': 20, '22': 21,
{'일': 0, '월': 1, '화': 2, '목': 3, '수': 4, '토': 5, '금': 6}
{'비포장': 0, '포장': 1}
{'해빙': 0, '침수': 1, '적설': 2, '서리/결빙': 3, '기타': 4, '젖음/습기': 5, '건조': 6}
{'안개': 0, '눈': 1, '기타/불명': 2, '흐림': 3, '비': 4, '맑음': 5}
{'철길건널목': 0, '불명': 1, '기타': 2, '기타/불명': 3, '교차로': 4, '단일로': 5}
{'철길건널목': 0, '불명': 1, '지하차도(도로)내': 2, '고가도로위': 3, '터널안': 4, '교차로횡단보도내': 5, '교량위': 6, '기타': 7, '횡단보도부근': 8, '기타/불명': 9, '횡단보도상': 10, '교차로부근': 11,
```

'발생시간', '요일', '노면상태', '기상상태', '도로형태' 범주형 변수에 대하여 정수형 변수로 라벨링하는 과정을 다음과 같이 각 변수에 대해 그룹화하여 '사고건수'의 합계를 sort_values()시켜,
'사고건수 ' 의 합이 작은 변수일수록 0에 가깝게, 합이 큰 변수일수록 숫자가 크도록,
오름차순으로 정렬하는 방식으로 재라벨링 하였습니다.

재라벨링 및 타 지역 교통사고 예측 모델 검증 작업

```
col = ['요일', '노면상태_대분류', '노면상태', '기상상태']
loc_data_seoul = new_data_seoul.groupby(['발생년도', '발생월', '발생일자', '도로형태', '도로형태_대분류']).mean()[col]
loc_data_seoul['사고건수']=new_data_seoul.groupby(['발생년도', '발생월', '발생일자', '도로형태', '도로형태_대분류']).sum()['사고건수']
loc_data_seoul['사고건수표준화'] = (loc_data_seoul['사고건수']-loc_data_seoul['사고건수'].mean()) / loc_data_seoul['사고건수'].std()
loc_data_seoul['사고건수최대최소화'] = (loc_data_seoul['사고건수']-loc_data_seoul['사고건수'].min()) / (loc_data_seoul['사고건수'].max() - loc_data_seoul['사고건수'].min())
loc_data_seoul = loc_data_seoul.reset_index(drop=False)
loc_data_seoul
```

	발생년도	발생월	발생일자	도로형태	도로형태_대분류	요일	노면상태_대분류	노면상태	기상상태	사고건수	사고건수표준화	사고건수최대최소화
0	2010	1	1	4	5	6.0	1.0	6.000000	5.000000	1	-0.778504	0.000000
1	2010	1	1	6	5	6.0	1.0	4.666667	4.333333	3	-0.673434	0.019417
2	2010	1	1	8	5	6.0	1.0	3.000000	3.000000	1	-0.778504	0.000000
3	2010	1	1	9	3	6.0	1.0	6.000000	5.000000	2	-0.725969	0.009709
4	2010	1	1	10	5	6.0	1.0	5.750000	5.000000	4	-0.620899	0.029126
...	...	...	...	...	...	...	...	...	...	...	...	...
22898	2018	12	31	5	4	1.0	1.0	6.000000	5.000000	5	-0.568364	0.038835
22899	2018	12	31	7	2	1.0	1.0	6.000000	5.000000	5	-0.568364	0.038835
22900	2018	12	31	11	4	1.0	1.0	5.894737	5.000000	19	0.167127	0.174757
22901	2018	12	31	12	4	1.0	1.0	5.900000	5.000000	20	0.219662	0.184466
22902	2018	12	31	13	5	1.0	1.0	6.000000	4.877193	57	2.163458	0.543689

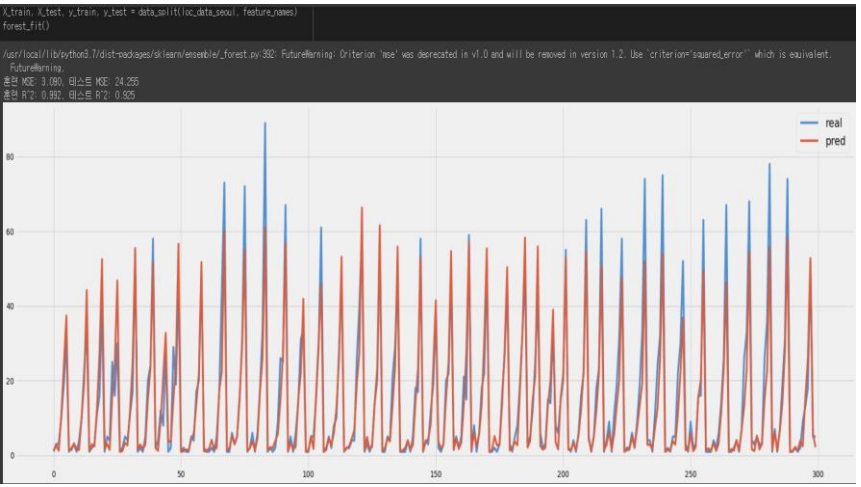
22903 rows × 12 columns

재라벨링된 데이터셋으로, 각각 발생년, 월, 일, 도로형태에 대해 그룹화하여,

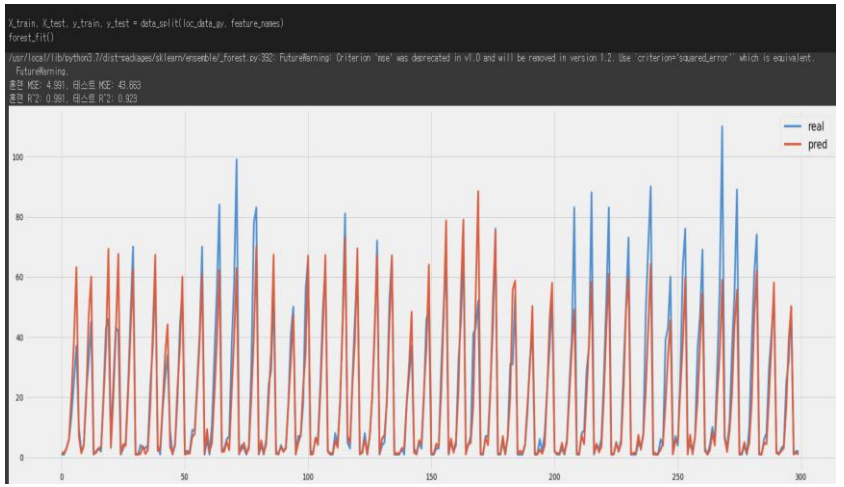
가변적인 변수인 '요일', '노면상태', '기상상태' 에 대하여 mean() 함수를 적용하였으며,

각 지역의 교통사고 데이터를 서울특별시 모델에 일반화 하기 위해, data_split1 , data_split2 함수를 사용할 목적으로, 사고건수를 표준화, 최대최소화 시킨 변수 또한 추가하였습니다.

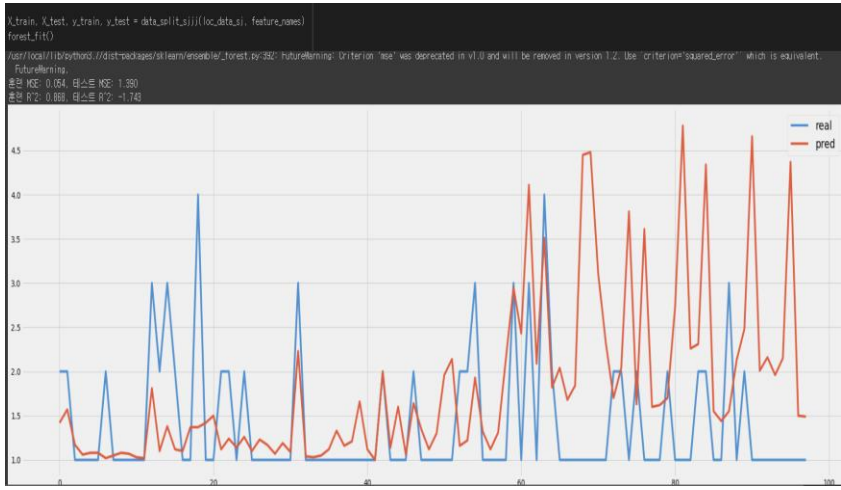
각 지역별로 훈련, 테스트 데이터로 나누어 예측한 결과는 다음과 같습니다.



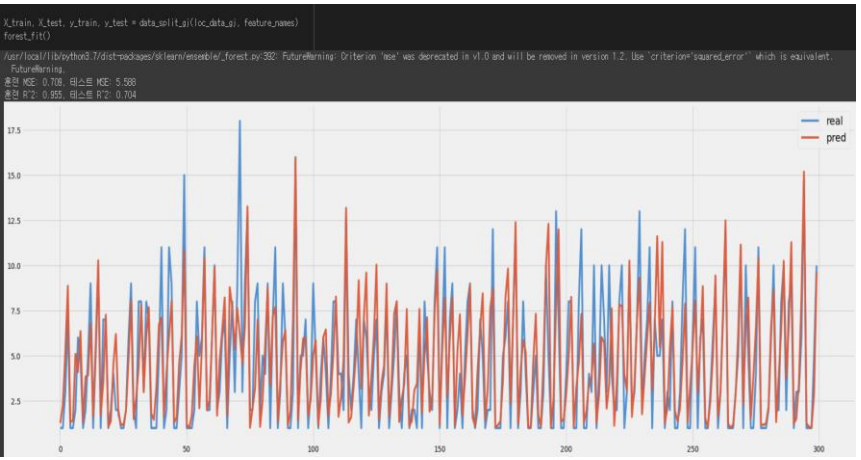
<서울시, 테스트 R^2 약 0.925>



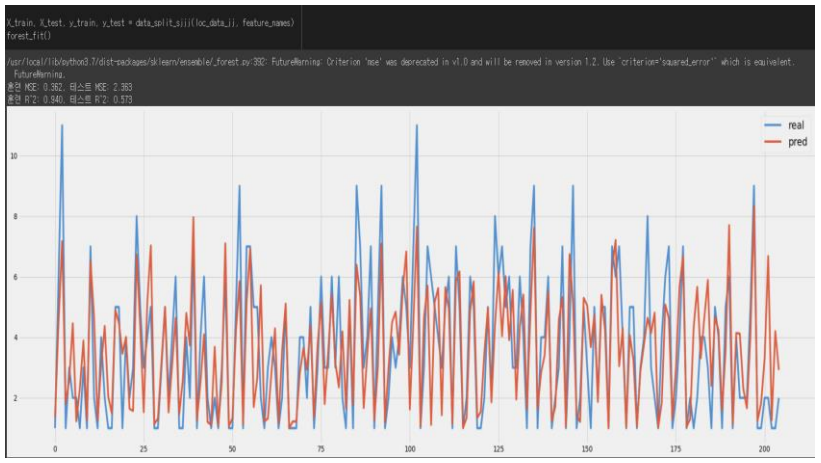
<경기도, 테스트 R^2 약 0.923>



<세종시, 테스트 R^2 매우 낮음>

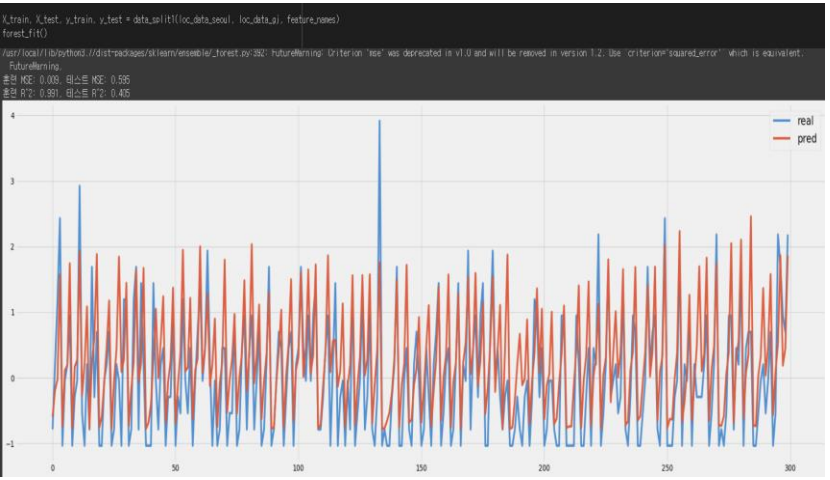


<광주시, 테스트 R^2 약 0.704>

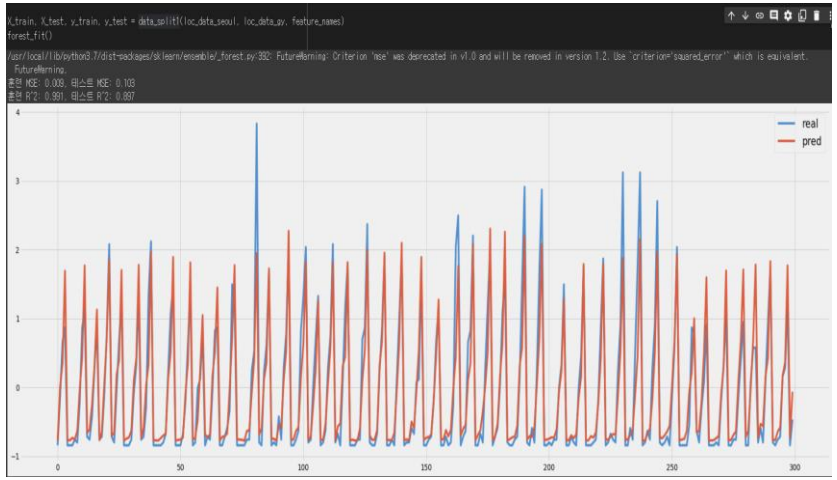


<제주도, 테스트 R^2 약 0.573>

서울특별시 데이터를 훈련, 각 지역의 데이터를 테스트 데이터로 사용하여 표준화한 모델

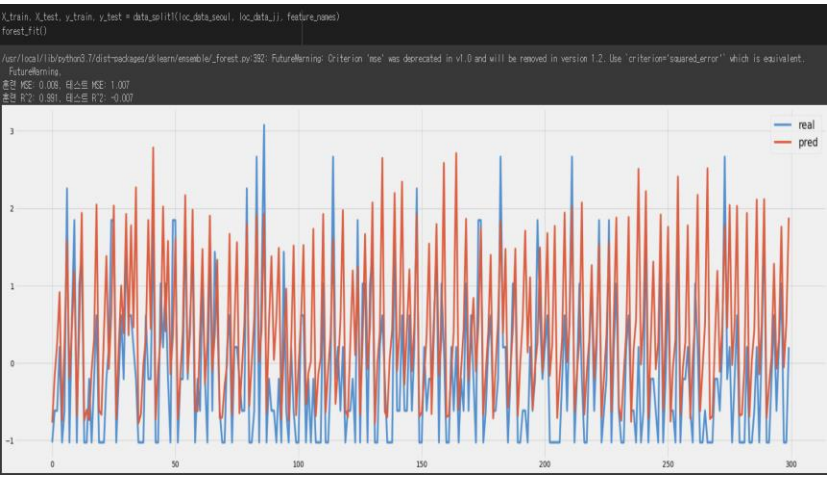


<광주시, 테스트 R^2 약 0.405>



<경기도, 테스트 R^2 약 0.897>

<세종시, 테스트 R^2 매우 낮음>



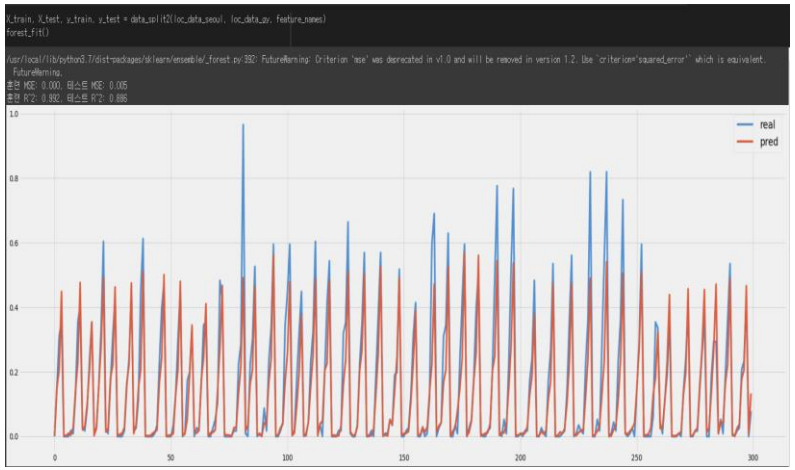
<제주도, 테스트 R^2 0에 가까움>

경기도의 경우, 서울특별시 데이터로 예측한 모델을 적용하였을 때 0.9에 가까운 높은 정확도가 나왔으며, 이는 서울특별시와 유동인구 수가 많고, 생활 패턴이 유사하기 때문이라고 예측해볼 수 있으며,

세종시의 경우에는 각 그룹별로 사고건수가 1~5의 매우 작은 값을 지니기 때문에, 경우의 수가 너무 적어서 일반화가 되지 않는 것으로 생각할 수 있었습니다.

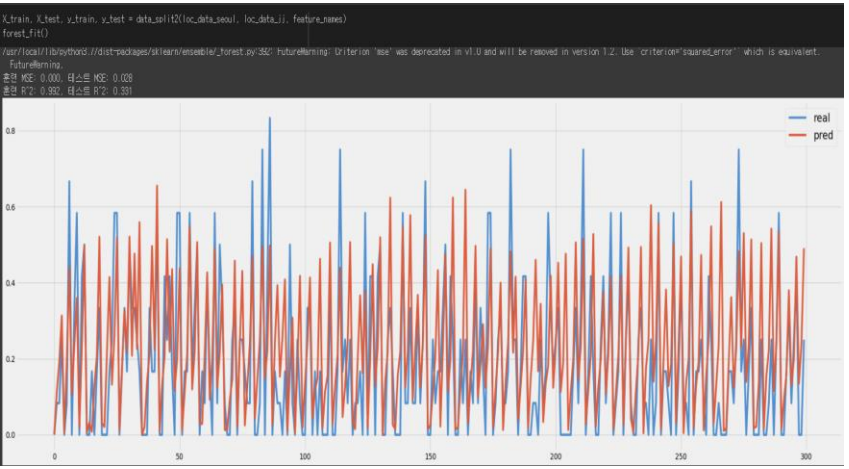
서울특별시 데이터를 훈련, 각 지역의 데이터를 테스트 데이터로 사용하여 최대-최소화한 모델

<광주시, 테스트 R^2 약 0.506>



<경기도, 테스트 R^2 약 0.886>

<세종시, 테스트 R^2 매우 낮음>



<제주도, 테스트 R^2 약 0.331>

표준화한 모델과 비교하였을 때, 경기도 데이터는 여전히 0.9에 비슷한 높은 R^2 정확도, 세종시 데이터는 음수 값에 해당하는 낮은 정확도를 보였으며,

광주, 제주도의 데이터는 표준화한 모델에서보다 최대-최소화한 모델에서 조금 더 높은 정확도를 보였음을 확인해볼 수 있으며, 이는 사고건수가 작은 수에서 최빈값을 가지는 성질을 반영할 수 있었기 때문에, 예측 정확도가 조금 높게 나온 것으로 생각해볼 수 있습니다.

서론

연마다 교통사고가 꾸준히 발생하고 있으며, 일부 지역에서는 교통사고가 일정 수준 이상으로 발생하는 장소를 선정하여 '개선사업' 대상지로 선정하여 현장 조사 및 원인 분석을 통해 개선점을 만듦으로서, 교통사고 발생 건수를 줄이는 등의 많은 노력을 기울이고 있습니다.

교통사고 다발 구역 개선사업, 개선했는데 여전히 사고위험 1위?...'무늬만 안전사업'

하지만, 경기도의 경우 잘 시행되지 않아, 오히려 사상자수가 증가하는 기사가 빈번히 올라오는 것을 확인해볼 수 있었으며,

교통사고 발생건수에 직접적으로 영향을 주는 변수가 무엇인지 모델을 통해 확인해보고자 교통사고 데이터를 직접 불러와서 모델링하는 작업을 통해 확인해보는 프로젝트를 진행하게 되었습니다.

결론

주로 교통사고 발생건수에 영향을 미치는 요인이 무엇인지 분석하고자 프로젝트를 진행하였는데, 랜덤포레스트로 특성을 분석한 결과, 발생건수에 가장 영향을 미치는 요인이 불변적인 변수로서는 도로 형태, 가변적인 변수로서는 날씨 및 노면상태의 데이터임을 확인할 수 있었습니다.

다른 지역의 교통사고 데이터에도 위의 모델을 일반화한 결과, 유동인구 수가 많은 경기도 지역에서 교통사고 발생건수의 분포가 서울시와 가장 유사한 패턴을 보인 것을 확인할 수 있었습니다.

추후 작업으로, 서울시의 교통사고 모델을 적용하여 미래의 날씨 데이터를 받아와서 실제로 교통사고 발생 건수의 예측이 실제 발생 건수와 얼마나 비슷한 결과를 나타내는지를 확인해보는 작업으로, 모델을 검증해보고 싶습니다!