제주도 도로 교통량 예측 프로젝트

4조 이재영, 고석주
양효준, 지우근

2022.12.09

목차

01 Problem Definition

01 Problem Definition

제주 '드라이브 스루' 늘어 교통혼잡...교통안전조례 제정해야

'인구 70만 시대'... 제주도민들이 기뻐하지 못하는 이유

- 제주도내 주민등록인구는 2022기준 약 68만명으로 연 평균 1.3%정도 매년 증가하고있다.

- 외국인과 관광객까지 고려하면 전체 상주인구는 90만명 넘을 것으로 예상된다.

- 제주도민 증가와 외국인의 증가로 현재 제주도의 교통체증의 심각한 문제로 대두되고 있다.

출처 – 아이엠피터

01 Problem Definition

이런 현상에 따라 어느 도로가 더욱 침체되는지 어느 달에 더 혼잡할 지 예측하여 정부와 시민에게 단기 장기적인 정보를 제공할 수 있다.

01 Problem Definition

주최자 : 제주 테크노파크,

제주특별자치도

주관 : Dacon

02 EDA

2-1 데이터 정보 확인

	변수명	변수 설명
0	id	아이디
1	base_date	날짜
2	day_of_week	요일
3	base_hour	시간대
4	road_in_use	도로사용여부
5	lane_count	차로수
6	road_rating	도로등급
7	multi_linked	중용구간 여부
8	connect_code	연결로 코드
9	maximum_speed_limit	최고속도제한
10	weight_restricted	통과제한하중
11	height_restricted	통과제한높이
12	road_type	도로유형
13	start_latitude	시작지점의 위도
14	start_longitude	시작지점의 경도
15	start_turn_restricted	시작 지점의 회전제한 유무
16	end_latitude	도착지점의 위도
17	end_longitude	도착지점의 경도
18	end_turn_restricted	도작지점의 회전제한 유무
19	road_name	도로명
20	start_node_name	시작지점명
21	end_node_name	도착지점명
22	vehicle_restricted	통과제한차량
23	target	평균속도(km)

```
len(train)
```

4701217

수치형 데이터는 target인 평균 속도가 유일

숫자로 이루어진 데이터값이 보이지만 모두 범주형 데이터입니다.

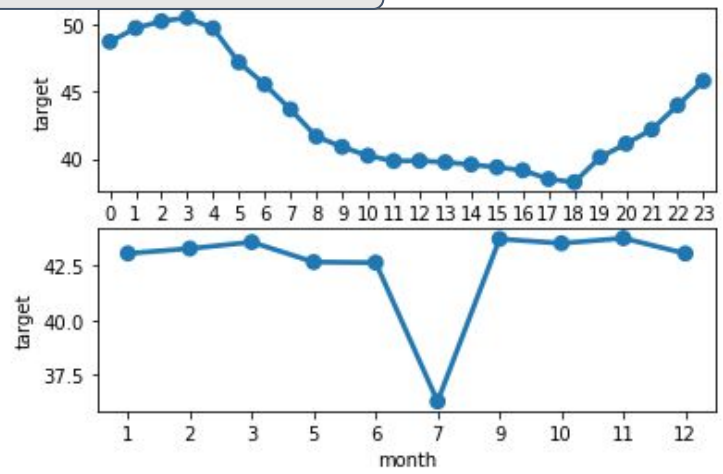
	0	1	2	3	4
id	TRAIN_0000000	TRAIN_0000001	TRAIN_0000002	TRAIN_0000003	TRAIN_0000004
base_date	20220623	20220728	20211010	20220311	20211005
day_of_week	목	목	일	금	화
base_hour	17	21	7	13	8
lane_count	1	2	2	2	2
road_rating	106	103	103	107	103
road_name	지방도1112호선	일반국도11호선	일반국도16호선	태평로	일반국도12호선
multi_linked	0	0	0	0	0
connect_code	0	0	0	0	0
maximum_speed_limit	60.0	60.0	80.0	50.0	80.0
vehicle_restricted	0.0	0.0	0.0	0.0	0.0
weight_restricted	32400.0	0.0	0.0	0.0	0.0
height_restricted	0.0	0.0	0.0	0.0	0.0
road_type	3	0	0	0	0
start_node_name	제3교래교	광양사거리	창고천교	남양리조트	애월샷시
start_latitude	33.427747	33.50073	33.279145	33.246081	33.462214
start_longitude	126.662612	126.529107	126.368598	126.567204	126.326551
start_turn_restricted	없음	있음	없음	없음	없음
end_node_name	제3교래교	KAL사거리	상창육교	서현주택	애월입구
end_latitude	33.427749	33.504811	33.280072	33.245565	33.462677
end_longitude	126.662335	126.52624	126.362147	126.566228	126.330152
end_turn_restricted	없음	없음	없음	없음	없음
target	52.0	30.0	61.0	20.0	38.0

2-1 데이터 정보 확인

train.groupby(['base_date'])['target'].size()

```
base_date
20210901    19722
20210902    18809
20210903    19880
20210904    17998
20210905    17836
            ...
20220727     9195
20220728     7601
20220729     5138
20220730     1845
20220731     5539
Name: target, Length: 281, dtype: int64
```

일별, 월별 평균 속도



권역별로 구분하여 변수 추가

'road_rating'

<표 2-9> 도로 등급 분류

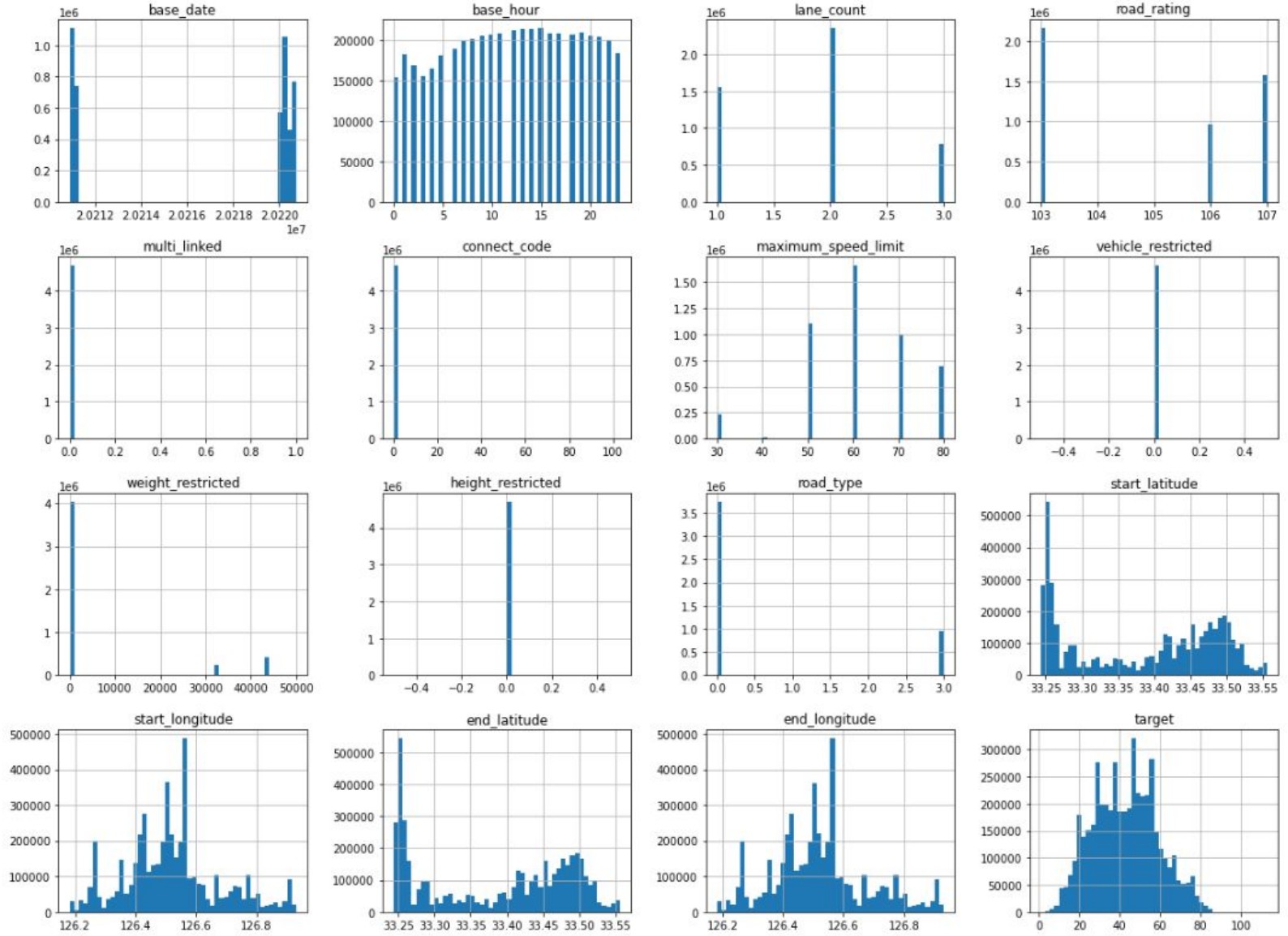
구분	도로 등급
101	고속도로
102	도시고속화도로
103	일반국도
104	특별광역시도
105	국가지원지방도
106	지방도
107	시도·군도
108	고속도로 연결램프

'connect_code'

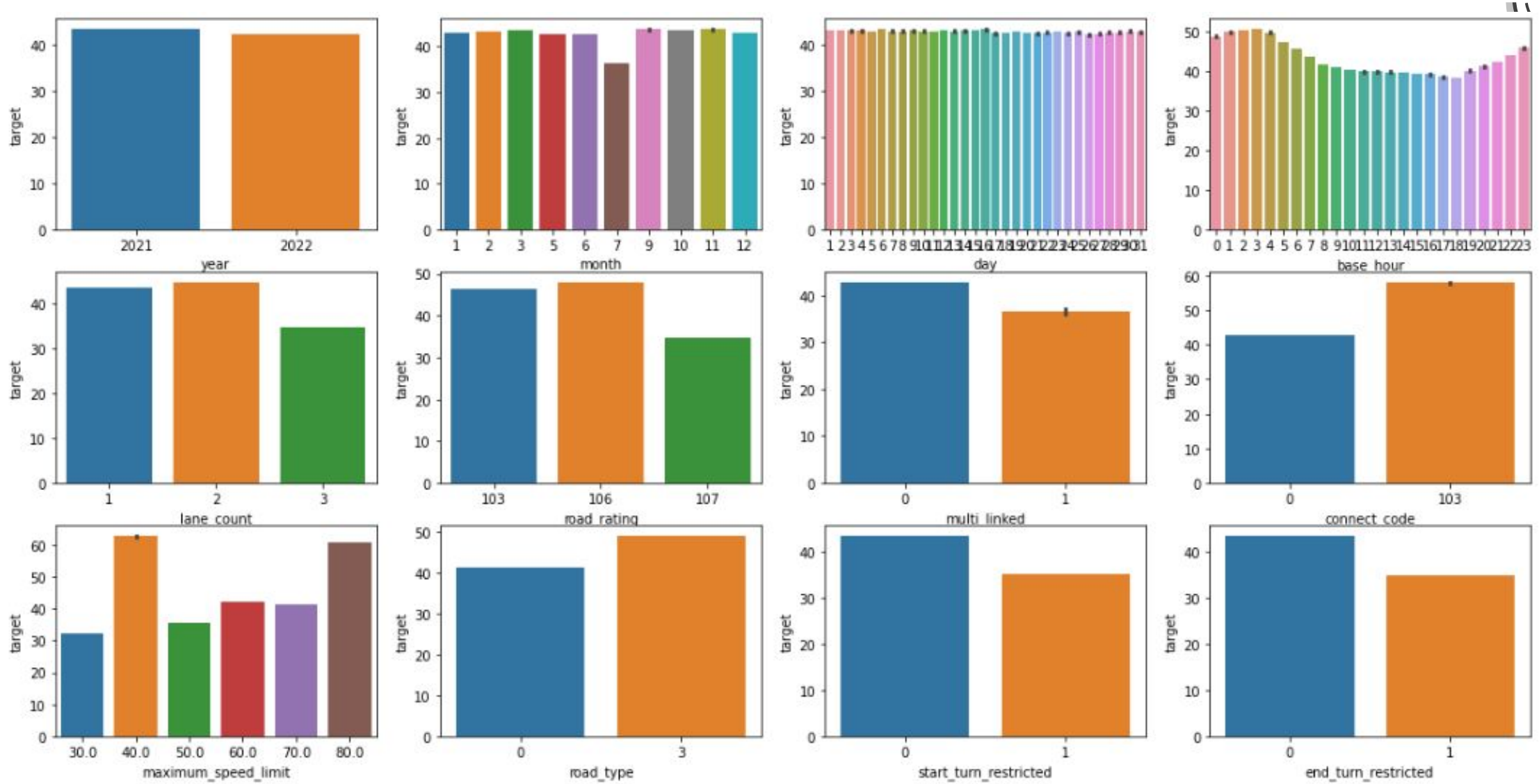
CONNECT (연결로코드)					
000	연결로 아님	103	일반국도 연결로	106	지방도 연결로
101	고속국도 연결로	104	특별광역시도 연결로	107	시군도 연결로
102	도시고속국도 연결로	105	국가지원지방도 연결로	108	기타도로 연결로

2-1 데이터 정보 확인

유일한 값을 가진 컬럼은 삭제를 해야겠다고 판단



2-1 데이터 정보 확인

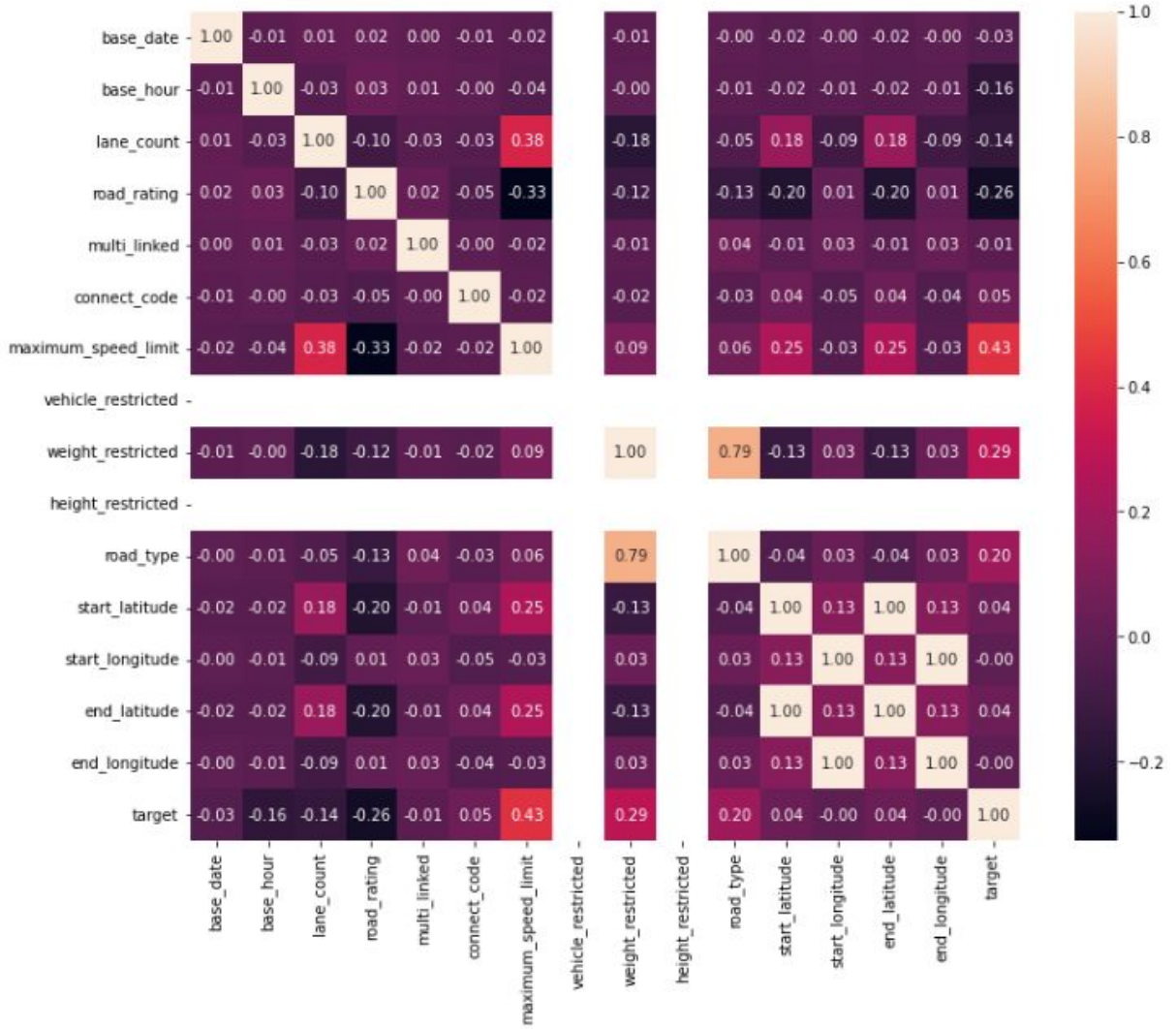

target
year
2021
2022
month
day
base_hour
lane_count
road_rating
multi_linked
connect_code
maximum_speed_limit
30.0
40.0
50.0
60.0
70.0
80.0
road_type
start_turn_restricted
end_turn_restricted

2-1 데이터 정보 확인

```
column_names = train.columns.values.tolist()

for i in column_names:
    print(f'{i} = {train[i].unique()}')
```

```
id = 4701217
base_date = 281
day_of_week = 7
base_hour = 24
lane_count = 3
road_rating = 3
road_name = 61
multi_linked = 2
connect_code = 2
maximum_speed_limit = 6
vehicle_restricted = 1
weight_restricted = 4
height_restricted = 1
road_type = 2
start_node_name = 487
start_latitude = 586
start_longitude = 586
start_turn_restricted = 2
end_node_name = 487
end_latitude = 586
end_longitude = 586
end_turn_restricted = 2
target = 102
```

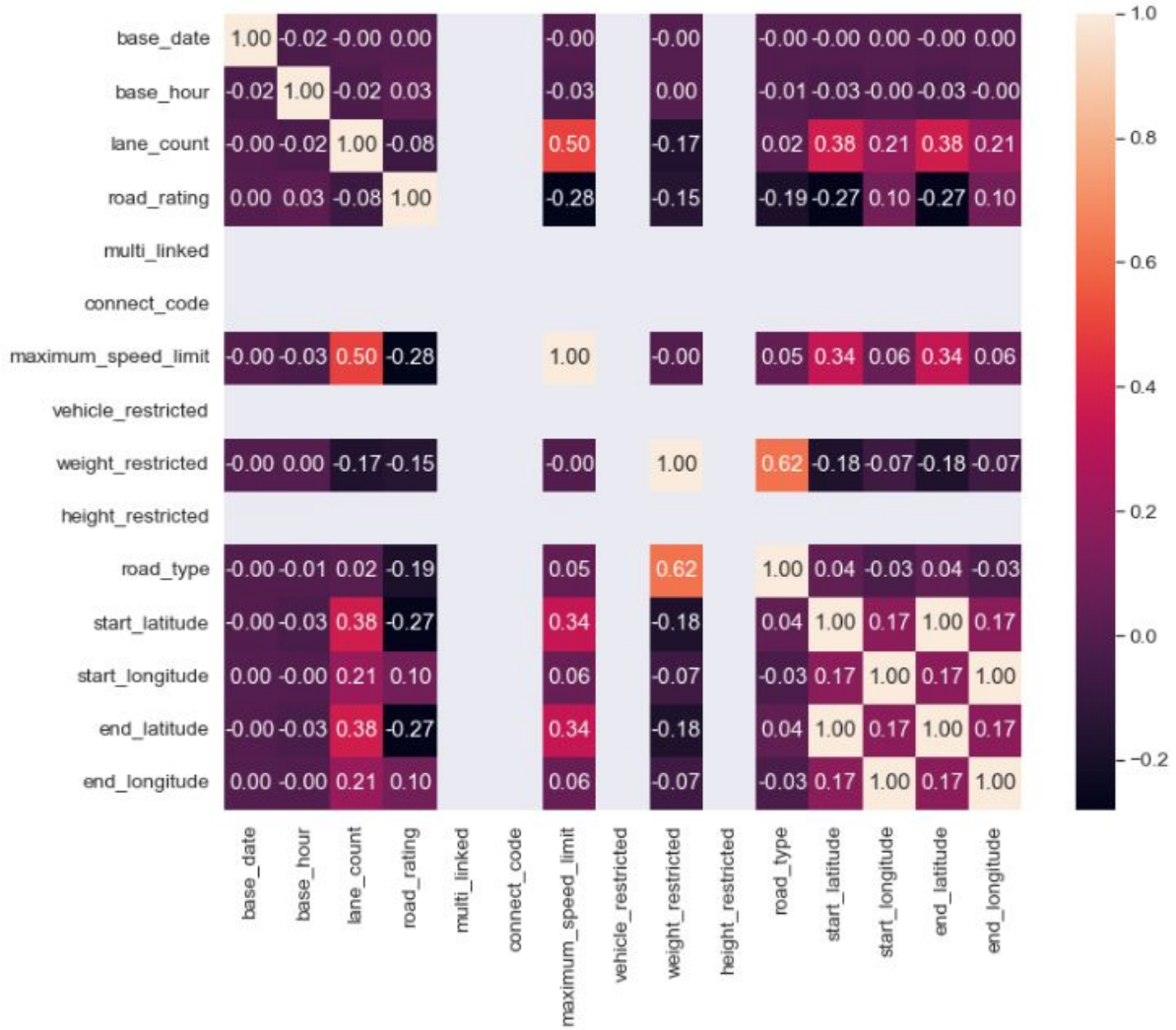
train 데이터 상관관계



2-2 test 데이터에서 이상한 점 발견

'connect_code', 'multi_linked' 컬럼

test 데이터 상관관계



2-3 ‘connect_code’, ‘multi_linked’ 컬럼 확인

train.connect_code.value_counts()

```
0      4689075
103      12142
Name: connect_code, dtype: int64
```

train.multi_linked.value_counts()

```
0    4698978
1       2239
Name: multi_linked, dtype: int64
```

```
print(train.groupby(connect_code)[‘target’].mean()
print(train.groupby(connect_code)[‘target’].std()
```

```
connect_code
0      42.749191
103    57.947044
Name: target, dtype: float32
connect_code
0      15.949711
103     9.075532
Name: target, dtype: float64
```

```
print(train.groupby(‘multi_linked’)[‘target’].mean()
print(train.groupby(‘multi_linked’)[‘target’].std()
```

```
multi_linked
0    42.791370
1    36.642696
Name: target, dtype: float32
multi_linked
0    15.954885
1    13.661950
Name: target, dtype: float64
```

2-3 'road_name'에서의 이상한 값 발견

길 이름이 '-'으로 표시된 것을 발견

train['road_name'].unique()

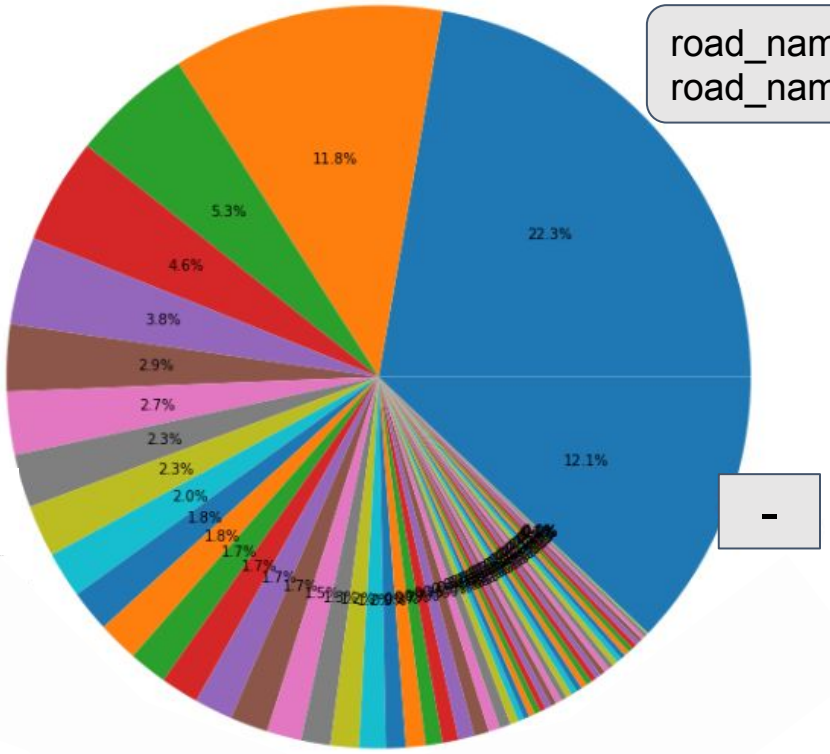
```
array(['지방도1112호선', '일반국도11호선', '일반국도16호선', '태평로', '일반국도12호선', '경찰로', '-',
       '외도천교', '일반국도99호선', '중정로', '번영로', '연동로', '중산간서로', '지방도1118호선',
       '새서귀로', '지방도1115호선', '지방도1132호선', '어시천교', '지방도1120호선', '삼무로',
       '애조로', '지방도1116호선', '일반국도95호선', '동부관광도로', '동홍로', '지방도97호선', '중문
로',
       '연삼로', '중앙로', '산서로', '지방도1117호선', '연북로', '남조로', '지방도1119호선', '동문로',
       '한천로', '삼봉로', '고평교', '연북2교', '관광단지로', '권학로', '시청로', '신대로', '서사로',
       '관덕로', '관광단지1로', '신산로', '관광단지2로', '신광로', '지방도1136호선', '첨단로',
       '제2거로교', '시민광장로', '임항로', '수영장길', '애원로', '삼성로', '일주동로', '호서중앙로',
       '아봉로', '호근로'], dtype=object)
```

train['road_name'].value_counts()

```
일반국도12호선     1046092
-             569463
일반국도16호선      554510
일반국도95호선      248181
일반국도11호선      215701
                ...
애원로              7718
아봉로              7342
남조로              6813
호서중앙로            2819
호근로               587
Name: road_name, Length: 61, dtype: int64
```

2-3 'road_name'에서의 이상한 값 발견

```
road_name_null.plot(kind = 'pie', autopct = '%.11%%', figsize=(16, 12))
```




```
road_name_null = train.groupby([train['road_name'] == '-'])['road_name'].value_counts()
road_name_null
```

```
road_name  road_name
False      일반국도12호선      1046092
           일반국도16호선       554510
           일반국도95호선       248181
           일반국도11호선       215701
           지방도1132호선       179200
                              ...
           아봉로                7342
           남조로                6813
           호서중앙로             2819
           호근로                 587
True       -                  569463
Name: road_name, Length: 61, dtype: int64
```

03 Data Preprocessing

데이터 전처리 목록

3-1 컬럼 삭제

3-2 ‘distance’, ‘airport_distance’ 컬럼 생성

3-3 ‘center_start’, ‘center_end’ 컬럼 생성

3-4 Target Encoding

3-5 결측값 제거

3-6 ‘season’ 컬럼 추가

3-7 ‘work_or_rest_or_other’ 컬럼 추가

3-8 ‘target’ 이상치 제거

3-9 label Encoding

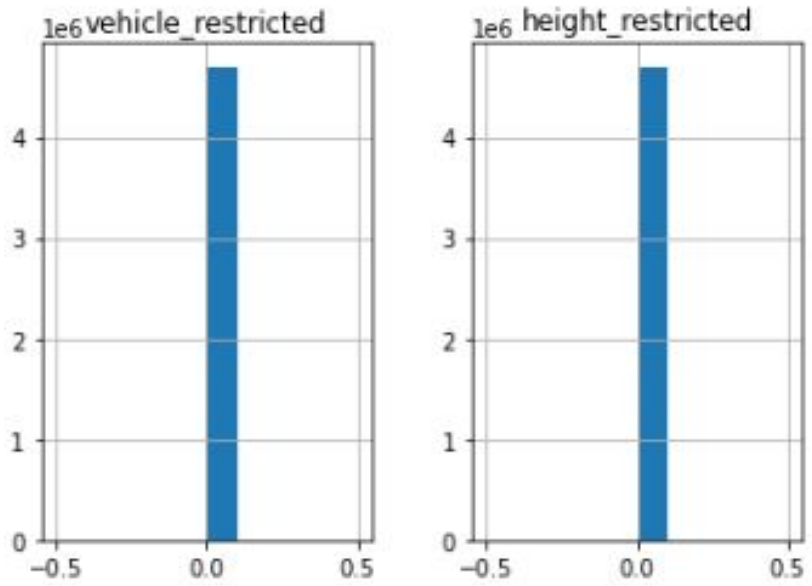
3 – 1 컬럼 삭제

하나의 값만 포함하고 있는 컬럼 삭제

```
train_desc = train.describe().transpose()
train_desc[train_desc['std']==0].index
```

```
Index(['vehicle_restricted', 'height_restricted'], dtype='object')
```

```
train_desc = train.describe().transpose()
train_desc[train_desc['std']==0].index
```



3-2 'distance', 'airport_distance' 컬럼 생성

경도 위도를 이용한 거리 파생변수 생성

시작 지점과 끝 지점 사이의 거리

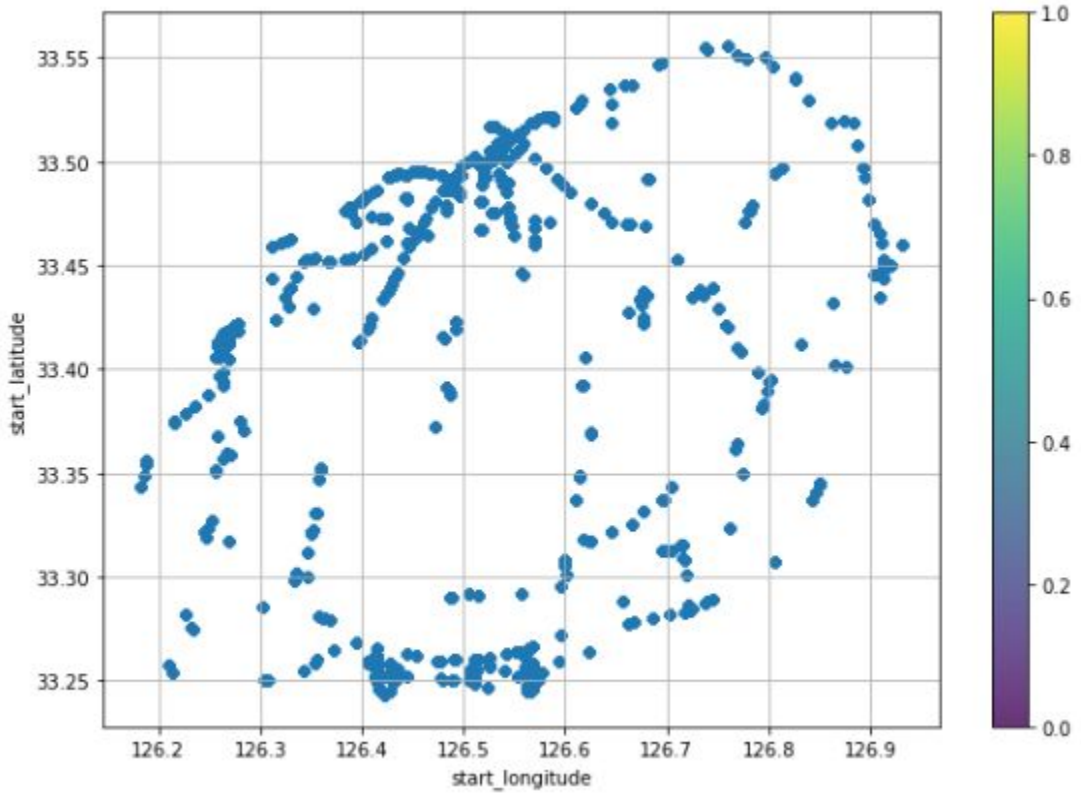
```
ttps://www.kaggle.com/code/speedoheck/calculate-distance-with-geo-coordinates/notebo
from math import radians, cos, sin, asin, sqrt

def haversine(row):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1 = row['start_longitude']
    lat1 = row['start_latitude']
    lon2 = row['end_longitude']
    lat2 = row['end_latitude']

    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return km

rain['distance'] = train.apply(haversine, axis=1)
['distance'] = test.apply(haversine, axis=1)
```

제주 공항으로부터 거리

```
 haversine_airport(row):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1 = 126.4913534
    lat1 = 33.5104135
    lon2 = (row['start_longitude'] + row['end_longitude']) / 2
    lat2 = (row['start_latitude'] + row['end_latitude']) / 2

    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return km

rain['airport_distance'] = train.apply(haversine_airport, axis=1)
t['airport_distance'] = test.apply(haversine_airport, axis=1)
```

시작지점 경도, 위도 시각화

```
train.plot(kind='scatter', x='start_longitude', y='start_latitude', alpha=0.8, grid=True,
           cmap='jet', colorbar=True, figsize=(10, 7), sharex=False)
```



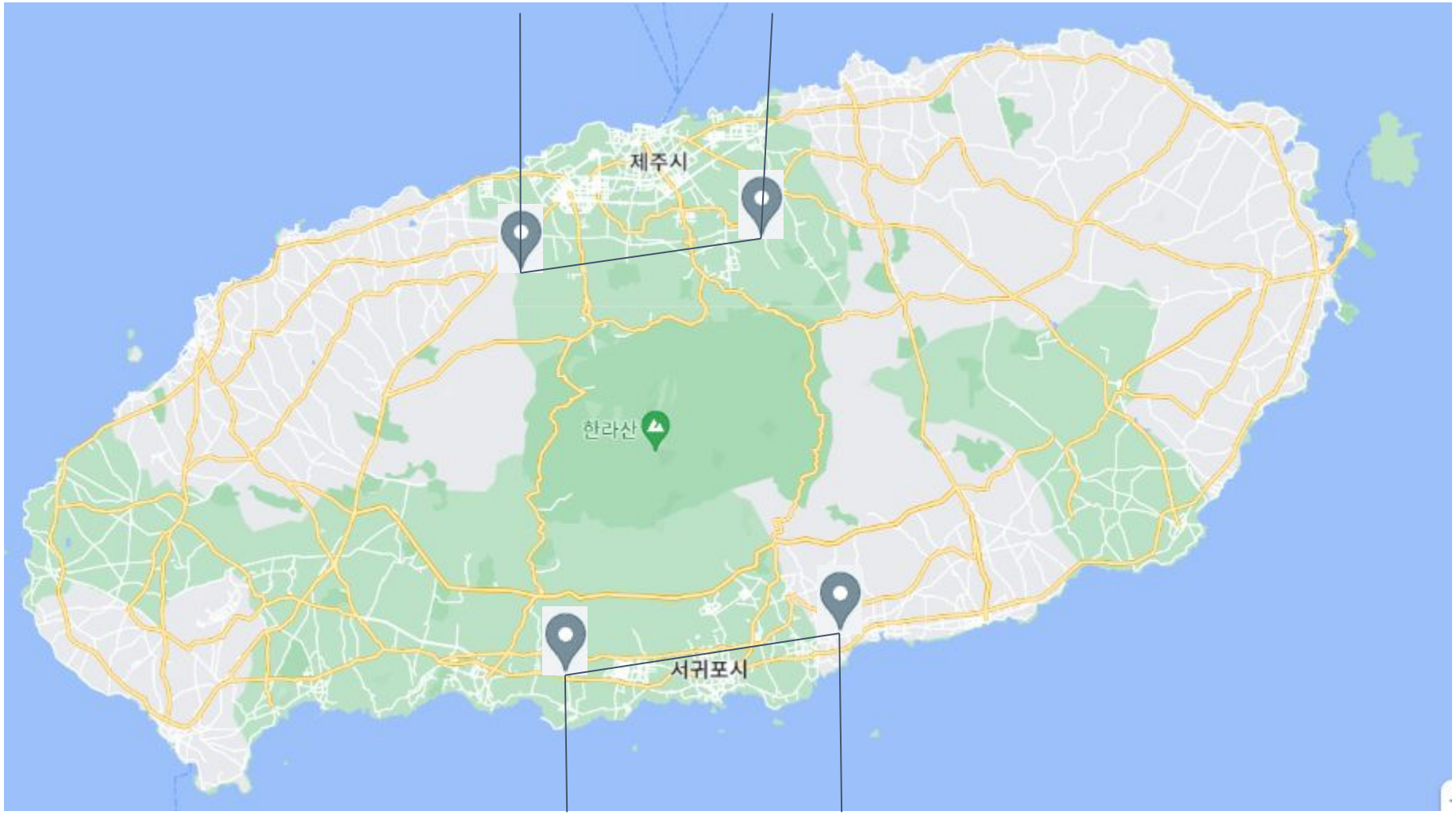
3-3 'center_start', 'center_end' 컬럼 생성

권역별로 구분하여 변수 추가

```
# 출발지점 권역
mask_jj_start = (train['start_longitude'] > 126.4531517) & (train['start_longitude']< 126.5900257) & (train['start_latitude'] > 33.4670429)
mask_jj_end = (train['end_longitude'] > 126.4531517) & (train['end_longitude']< 126.5900257) & (train['end_latitude'] > 33.4670429)

mask_sgp_start = (train['start_longitude'] > 126.3972753) & (train['start_longitude']< 126.6076604) & (train['start_latitude'] < 33.2686052)
mask_sgp_end = (train['end_longitude'] > 126.3972753) & (train['end_longitude']< 126.6076604) & (train['end_latitude'] < 33.2686052)
```

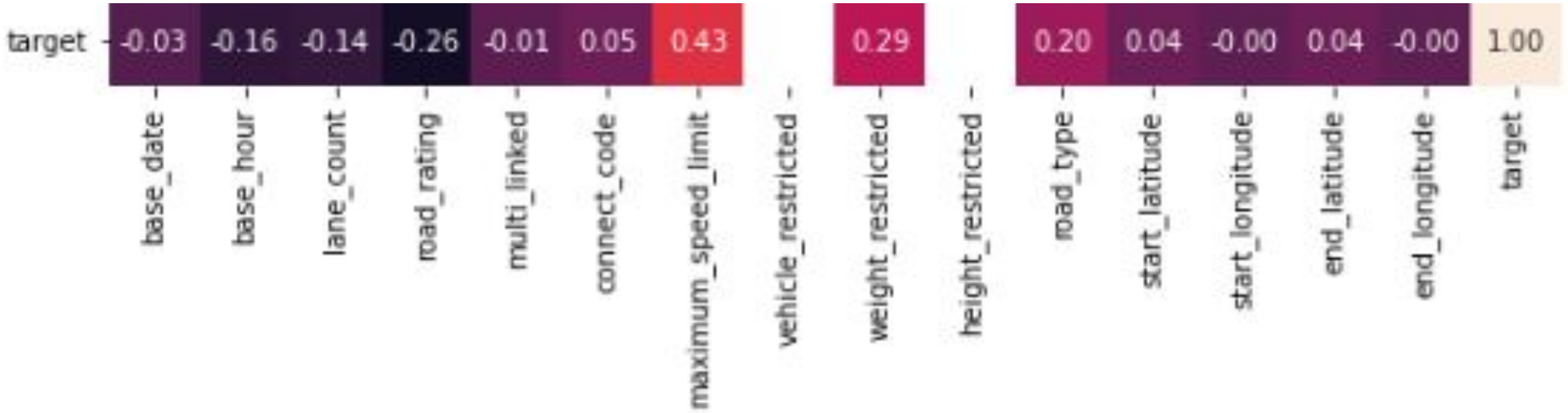
```
train['center_start'] = 0
test['center_start'] = 0
train.loc[mask_jj_start, 'center_start'] = 1
train.loc[mask_sgp_start, 'center_start'] = 2
test.loc[mask_jj_start, 'center_start'] = 1
test.loc[mask_sgp_start, 'center_start'] = 2

train['center_end'] = 0
test['center_end'] = 0
train.loc[mask_jj_end, 'center_end'] = 1
train.loc[mask_sgp_end, 'center_end'] = 2
test.loc[mask_jj_end, 'center_end'] = 1
test.loc[mask_sgp_end, 'center_end'] = 2
```

표시된 범위 지도를 통한 시각화



3-4 Target Encoding

GPS 정보를 이용하여 road 구분 후 target encoding



- 상관계수가 높은 컬럼이 많지 않아 컬럼을 추가하기도 했지만 인코딩도 활용해야겠다고 판단
- One-Hot Encoding을 할 경우 새로운 컬럼이 너무 많이 생기게 되어 High Cardinality 피처가 모델을 너무 불균형하게 만들어 column Sampling 과정에 안 좋은 영향을 끼침
- 그래서 Target Encoding 방식 이용

3-4 Target Encoding

GPS 정보를 이용하여 road 구분 후 target encoding

장점(label encoding과 비교했을 때)

- label encoding의 경우 1000개의 데이터면 1000개의 label이 생기는데 Mean target encoding은 카테고리 종류 수만큼 label이 생기므로 더욱 적은 split이 생기고 학습이 더욱 빠르게 이루어집니다.
- 인코딩된 label 값이 target 값과 관련된 의미를 가지게 되었으므로 less bias를 가집니다.

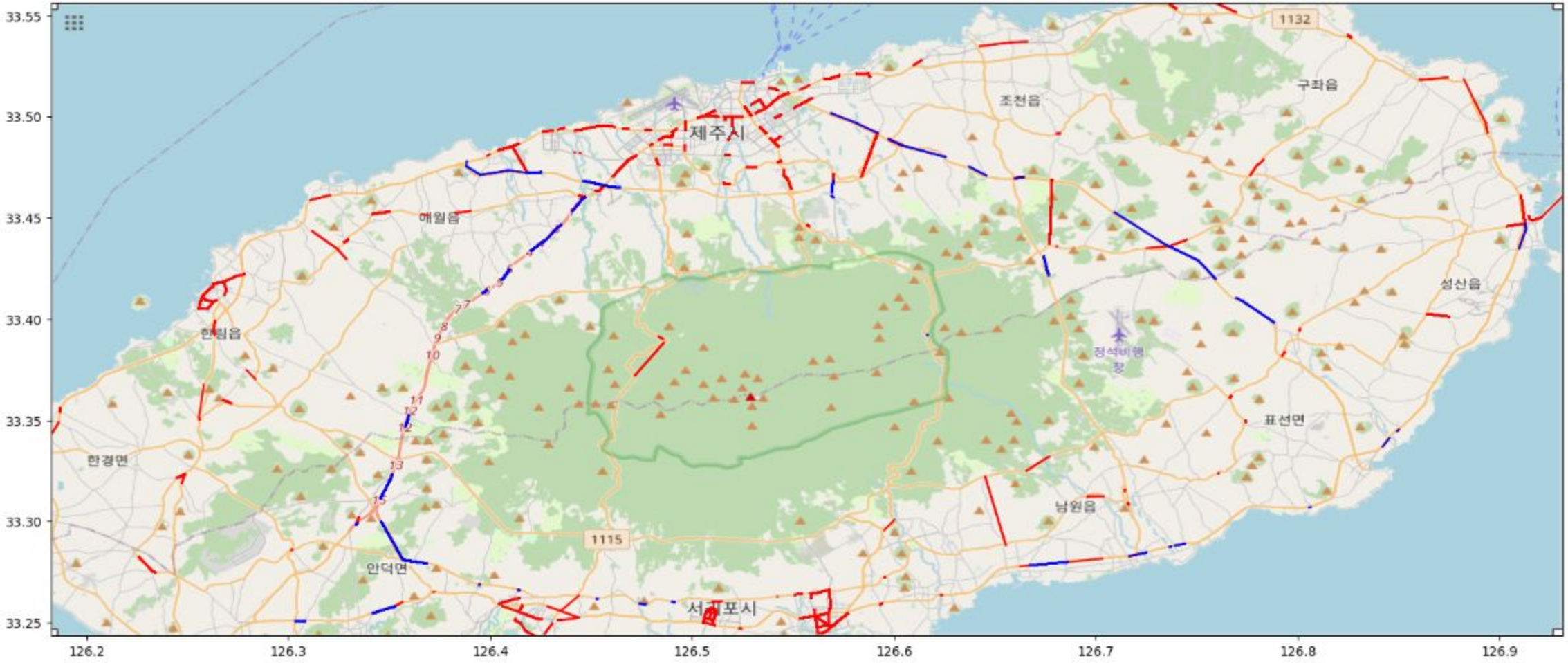
단점

- 구현과 검증이 까다롭습니다.
- Overfitting이 일어나기 쉽습니다.

	id	base_date	day_of_week	base_hour	lane_count	...	road_min	road_mean	road_max
0	TRAIN_0000000	20220623	목	17	1	...	16.0	51.756910	72.0
1	TRAIN_0000001	20220728	목	21	2	...	5.0	26.400712	59.0
2	TRAIN_0000002	20211010	일	7	2	...	32.0	59.101720	88.0
3	TRAIN_0000003	20220311	금	13	2	...	2.0	25.024923	51.0
4	TRAIN_0000004	20211005	화	8	2	...	25.0	39.873670	72.0

3-4 Target Encoding

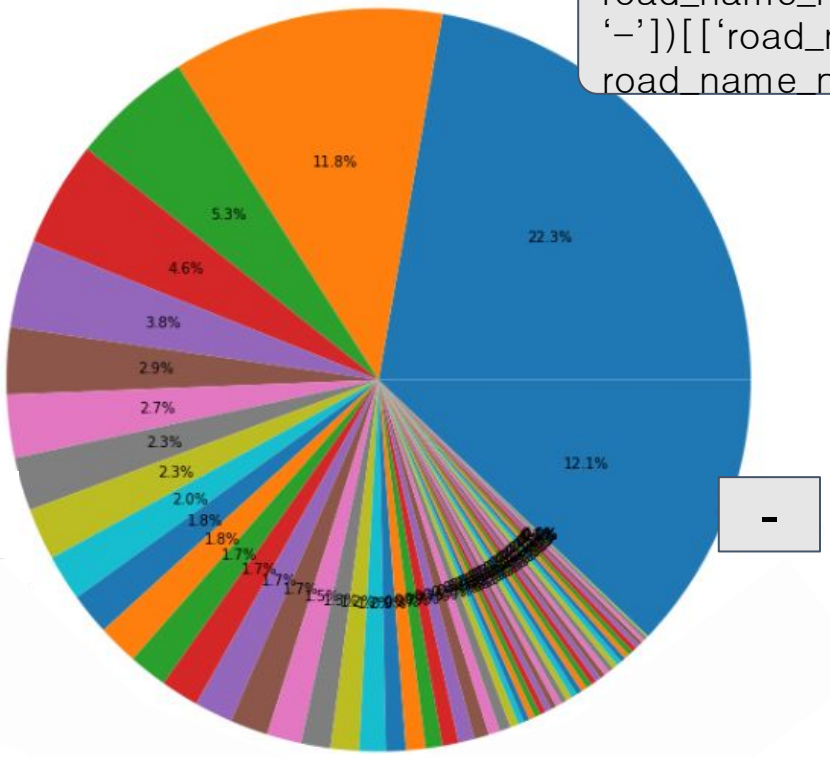
GPS 정보를 이용하여 road 구분 후 target encoding



3-5 결측값 제거

‘road_name’ 컬럼의 “-” (결측값) 제거

```
road_name_null.plot(kind = ‘pie’, autopct = ‘%.11%%’, figsize=(16, 12))
```




```
road_name_null = train.groupby([train[‘road_name’] ==
‘-’])[[‘road_name’].value_counts()
road_name_null
```

```
road_name  road_name
False      일반국도12호선      1046092
           일반국도16호선       554510
           일반국도95호선       248181
           일반국도11호선       215701
           지방도1132호선       179200
                              ...
           아봉로                7342
           남조로                6813
           호서중앙로            2819
           호근로                 587
True       -                  569463
Name: road_name, Length: 61, dtype: int64
```

3-5 결측값 제거

'road_name' 컬럼의 "-" (결측값) 제거

제거 전

```
print(len(train['road_name'] == "-"]))
```

569463

제거 후

```
print(len(train['road_name'] == "-"]))
```

201686

```
# 위도, 경도에 대해서 대체할 값 탐색
for i in train['end_latitude'].unique():
    if (len(train[(train['end_latitude'] == i)]['road_name'].value_counts())
!= 2):
        continue
    if '-' in train[(train['end_latitude'] ==
i)]['road_name'].value_counts().index:
        print("----------", i, '----------')
        print(train[(tain['end_latitude'] == i)]['road_name'].value_counts())
```

이런 방법으로 다른 road_rating, 시작지점, 끝지점의 이름까지 이용해서 결측치를 최대한 제거

```
---------- 33.47339 ----------
-          5338
일반국도12호선    5334
Name: road_name, dtype: int64
---------- 33.358358 ----------
-          4784
일반국도16호선    2251
Name: road_name, dtype: int64
---------- 33.412573 ----------
-      4389
월계교    4199
Name: road_name, dtype: int64
---------- 33.244882 ----------
-      5528
산서로    5415
Name: road_name, dtype: int64
---------- 33.322018 ----------
중문로    5187
-      2493
Name: road_name, dtype: int64
```

3-6 'season' 컬럼 추가

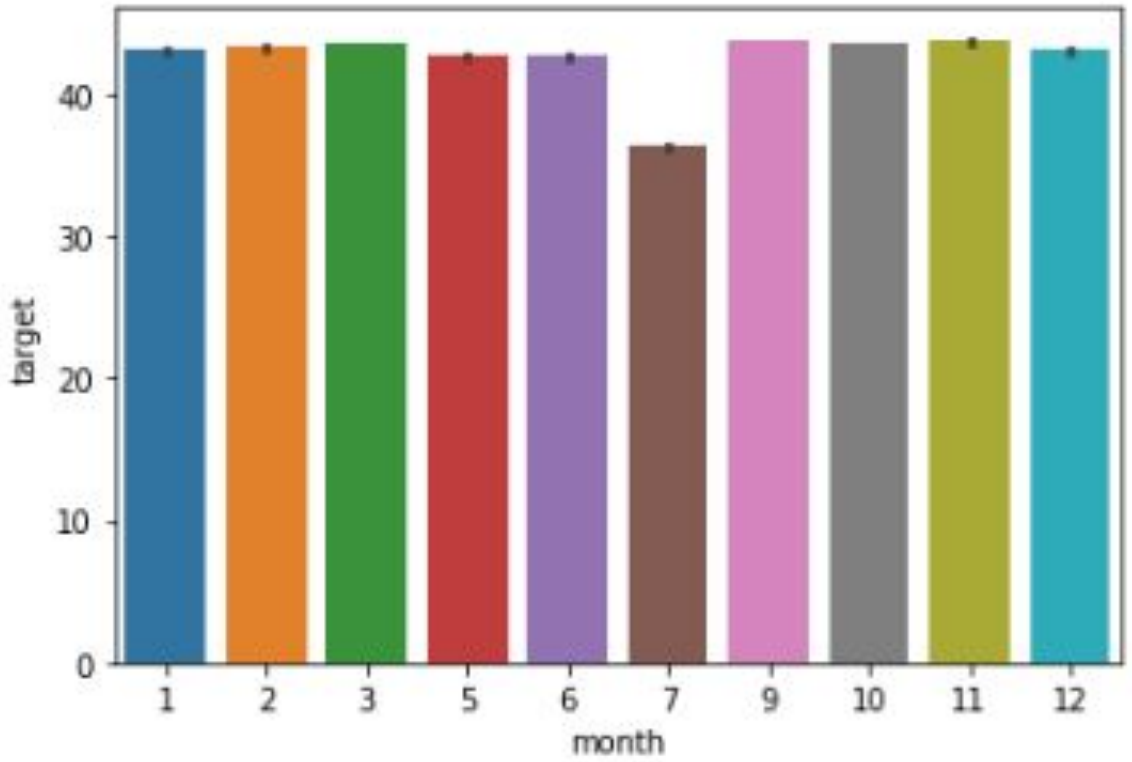
train 데이터에는 예측해내야 하는 8월의 데이터가 없는 점을 고려해서 7~9월인 값과 아닌 값으로 구분하여 컬럼 추가

```
def add_seson(x):
    if x == 1:
        season = 'not_7_to_9'
    elif x == 2:
        season = 'not_7_to_9'
    elif x == 3:
        season = 'not_7_to_9'
    elif x == 4:
        season = 'not_7_to_9'
    elif x == 5:
        season = 'not_7_to_9'
    elif x == 6:
        season = 'not_7_to_9'
    elif x == 7:
        season = 'seven_to_nine'
    elif x == 8:
        season = 'seven_to_nine'
```

```
    elif x == 9:
        season = 'seven_to_nine'
    elif x == 10:
        season = 'not_7_to_9'
    elif x == 11:
        season = 'not_7_to_9'
    elif x == 12:
        season = 'not_7_to_9'
    else:
        season = 'not_7_to_9'
    return season
```

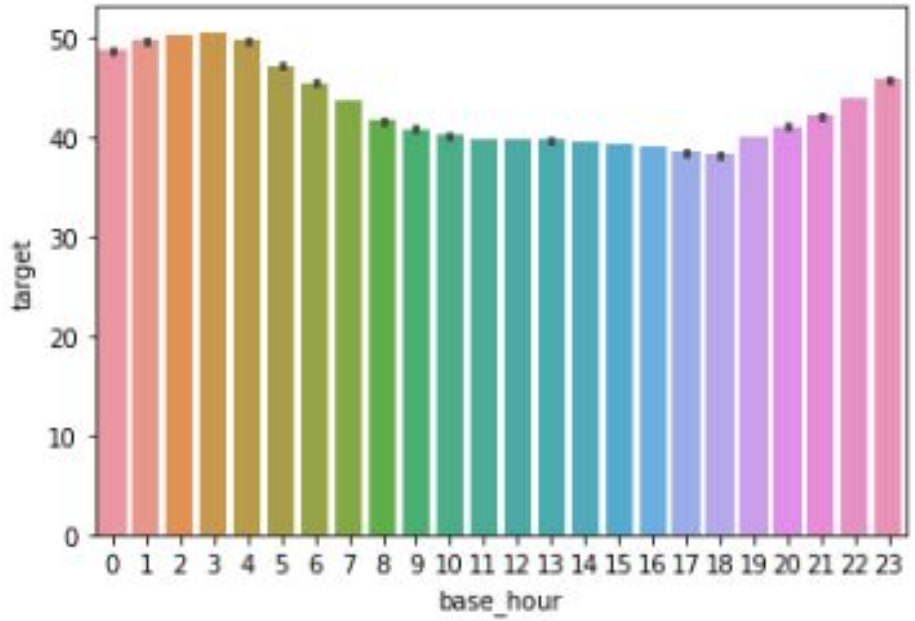
월별 평균 속도 시각화

```
sns.barplot(data=train, x='train', y='target')
```



3-7 'work_or_rest_or_other' 컬럼 추가

근무 시간이라고 판단되는 8~20시와 그 밖에 21~7시로 나누어 컬럼 추가

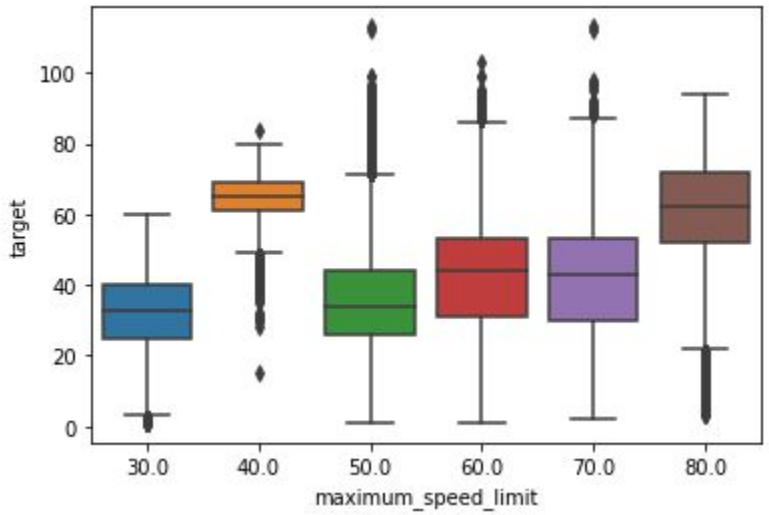


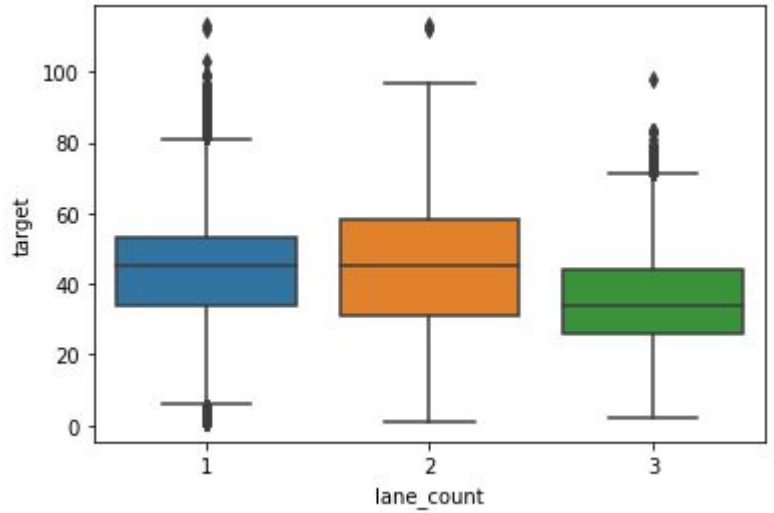

```
train.work_or_rest_or_other.value_counts()
```

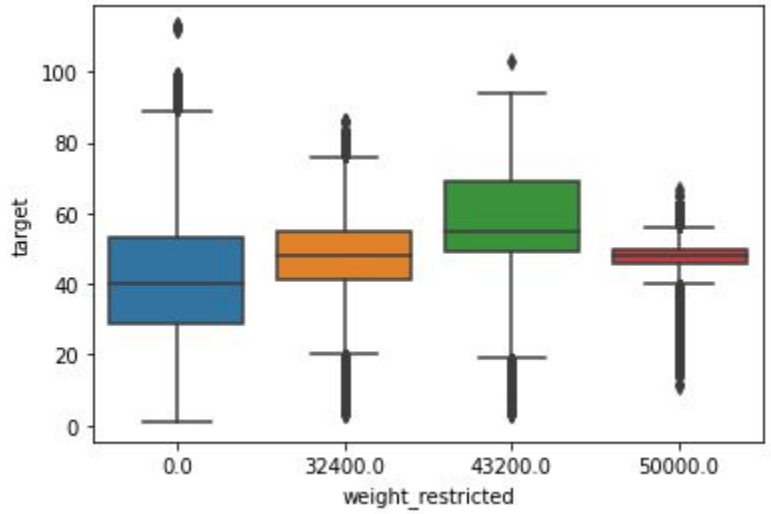
```
worktime    2716112
resttime    1985105
Name: work_or_rest_or_other, dtype: int64
```

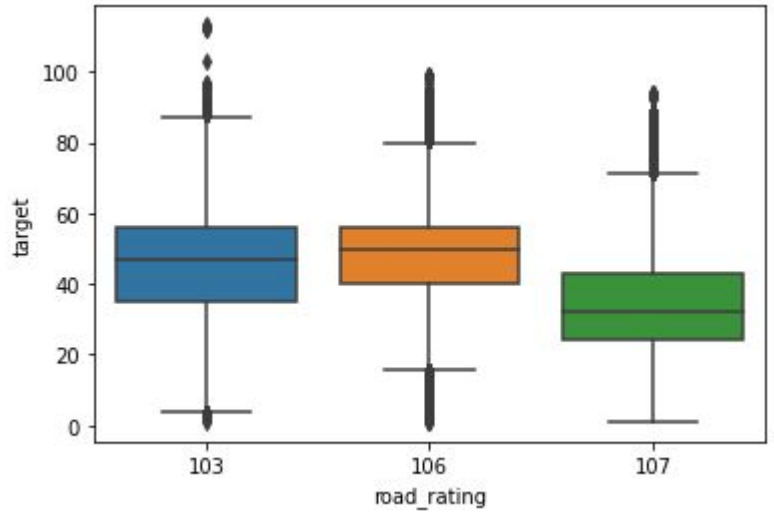
3-8 target 이상치 제거

평균 속도 100km 이상인 데이터 삭제









3-9 라벨 인코더

숫자가 아닌 값들 숫자로 변환시켜주기

	0	1	2	3	4
id	TRAIN_0000000	TRAIN_0000001	TRAIN_0000002	TRAIN_0000003	TRAIN_0000004
base_date	20220623	20220728	20211010	20220311	20211005
day_of_week	목	목	일	금	화
base_hour	17	21	7	13	8
lane_count	1	2	2	2	2
road_rating	106	103	103	107	103
road_name	지방도1112호선	일반국도11호선	일반국도16호선	태평로	일반국도12호선
multi_linked	0	0	0	0	0
connect_code	0	0	0	0	0
maximum_speed_limit	60.0	60.0	80.0	50.0	80.0
vehicle_restricted	0.0	0.0	0.0	0.0	0.0
weight_restricted	32400.0	0.0	0.0	0.0	0.0
height_restricted	0.0	0.0	0.0	0.0	0.0
road_type	3	0	0	0	0
start_node_name	제3교래교	광양사거리	창고천교	남양리조트	애월샷시
start_latitude	33.427747	33.50073	33.279145	33.246081	33.462214
start_longitude	126.662612	126.529107	126.368598	126.567204	126.326551
start_turn_restricted	없음	있음	없음	없음	없음
end_node_name	제3교래교	KAL사거리	상창육교	서현주택	애월입구
end_latitude	33.427749	33.504811	33.280072	33.245565	33.462677
end_longitude	126.662335	126.52624	126.362147	126.566228	126.330152
end_turn_restricted	없음	없음	없음	없음	없음
target	52.0	30.0	61.0	20.0	38.0

```
str_col = ['day_of_week', 'road_name',
'start_node_name', 'end_node_name',
'start_turn_restricted', 'end_turn_restricted',
'weight_restricted', 'road_rating', 'road_type',
'season', 'work_or_rest_or_other']

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

    for label in np.unique(test[i]):
        if label not in le.classes_:
            le.classes_ = np.append(le.classes_,
label)
    test[i] = le.transform(test[i])
```

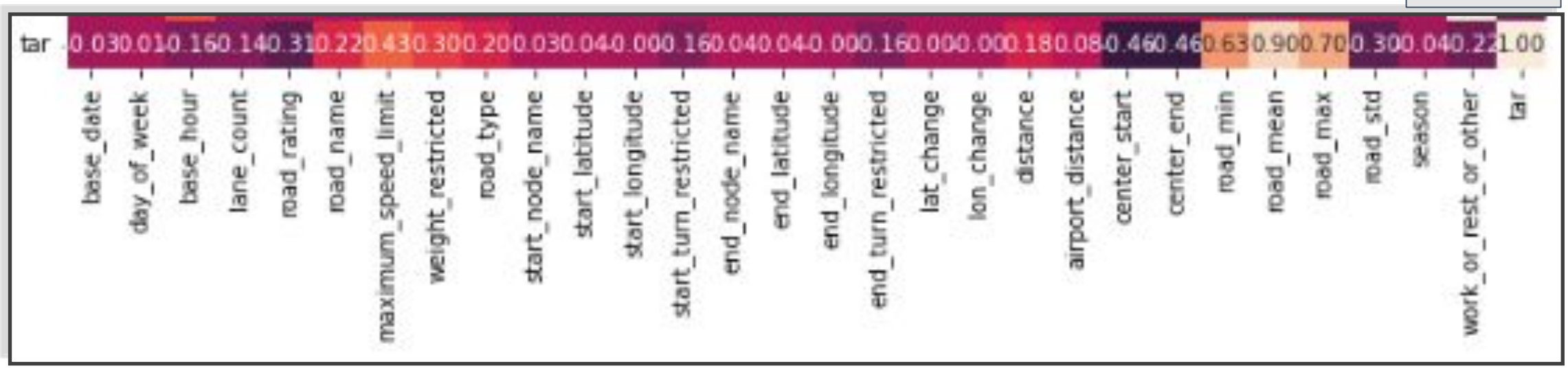
전처리 결과 확인

3-4 상관계수 확인

```
Y_train = train['target']
X_train = train.drop['id', 'multi_linked', 'connect_code', 'target'], axis = 1)
X_test = test.drop(['id', 'multi_linked', 'connect_code'], axis = 1)
```

```
X_train['tar'] = Y_train
train_corr = X_train.corr()
f, ax = plt.subplots(figsize=(16, 12))
sns.heatmap(train_corr, annot=True, fmt = '.2f', square=True)
```

컬럼 30개



04 Modeling

4-1 메모리 축소

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4701212 entries, 0 to 4701216
Data columns (total 34 columns):
 #   Column                 Dtype
---  ------                 -----
 0   id                     object
 1   base_date              int32
 2   day_of_week            int32
 3   base_hour              int64
 4   lane_count             int64
 5   road_rating            int64
 6   road_name              int32
 7   multi_linked           int64
 8   connect_code           int64
 9   maximum_speed_limit    float64
 10  weight_restricted      int64
 11  road_type              int64
 12  start_node_name        int32
 13  start_latitude         float64
 14  start_longitude        float64
 15  start_turn_restricted  int32
 16  end_node_name          int32
 17  end_latitude           float64
 18  end_longitude          float64
 19  end_turn_restricted    int32
 20  target                 float64
 21  lat_change             float64
 22  lon_change             float64
 23  distance               float64
 24  airport_distance       float64
 25  center_start           int64
 26  center_end             int64
 27  road_code              object
 28  road_min               float64
 29  road_mean              float64
 30  road_max               float64
 31  road_std               float64
 32  season                 int32
 33  work_or_rest_or_other  int32
dtypes: float64(14), int32(9), int64(9), object(2)
memory usage: 1.1+ GB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4701212 entries, 0 to 4701216
Data columns (total 34 columns):
 #   Column                 Dtype
---  ------                 -----
 0   id                     object
 1   base_date              int32
 2   day_of_week            int8
 3   base_hour              int8
 4   lane_count             int8
 5   road_rating            int8
 6   road_name              int8
 7   multi_linked           int64
 8   connect_code           int64
 9   maximum_speed_limit    int8
 10  weight_restricted      int8
 11  road_type              int8
 12  start_node_name        int16
 13  start_latitude         float64
 14  start_longitude        float64
 15  start_turn_restricted  int8
 16  end_node_name          int16
 17  end_latitude           float64
 18  end_longitude          float64
 19  end_turn_restricted    int8
 20  target                 float64
 21  lat_change             float64
 22  lon_change             float64
 23  distance               float64
 24  airport_distance       float64
 25  center_start           int64
 26  center_end             int64
 27  road_code              object
 28  road_min               float64
 29  road_mean              float64
 30  road_max               float64
 31  road_std               float64
 32  season                 int8
 33  work_or_rest_or_other  int8
dtypes: float64(13), int16(2), int32(1), int64(4), int8(12), object(2)
memory usage: 807.0+ MB
```

```
# 데이터 램 사용량을 감소시키기 위해 더이상 필요하지 않은 데이터 제거
del train
del test
gc.collect()
```

706

4-2 모델간 예측 성능 비교

LinearRegression

12.598237553430662

Ridge + GridSearch

```
12.598234970028102 {'alpha': 0.1}
12.598235059335535 {'alpha': 1}
12.598235608023254 {'alpha': 2}
12.598236540466395 {'alpha': 3}
12.598237781460107 {'alpha': 4}
12.598248959733015 {'alpha': 10}
12.598291948141792 {'alpha': 30}
12.598370854912703 {'alpha': 100}
12.598410514549569 {'alpha': 200}
12.598428116931377 {'alpha': 300}
12.59843812658374 {'alpha': 400}
12.598455664239928 {'alpha': 800}
12.598458019249284 {'alpha': 900}
12.59846004880126 {'alpha': 1000}
```

DecisionTreeRegressor

5.63397062152234

4-2 모델간 예측 성능 비교

RandomForestRegressor

4.47978887708306

Kfold + RandomForestRegressor + ExtraTreesRegressor

```
RandomForestRegressor mae : 2.9155

ExtraTreeRegressor mae : 2.9633

RandomForestRegressor mae : 2.9162

ExtraTreeRegressor mae : 2.9632

RandomForestRegressor mae : 2.9165

ExtraTreeRegressor mae : 2.9623

mean mae 2.9161
mean mae 2.9629
```

4-2 모델간 예측 성능 비교

RandomForestRegressor

4.47978887708306

Kfold + RandomForestRegressor + ExtraTreesRegressor

```
RandomForestRegressor mae : 2.9155

ExtraTreeRegressor mae : 2.9633

RandomForestRegressor mae : 2.9162

ExtraTreeRegressor mae : 2.9632

RandomForestRegressor mae : 2.9165

ExtraTreeRegressor mae : 2.9623

mean mae 2.9161
mean mae 2.9629
```

감사합니다