데이콘 데이터

제주도 도로 교통량 예측

비즈인포매틱스학과
2021100538
허지연

차례

1. 비즈니스 문제 정의

[제주시 교통량을 예측하는 AI 서비스]

교통 지원 정책 / 비즈니스 활용 가능

“제주시의 교통량을 분석함으로써 어느 시기에 어디에서 교통량이 급증하는지 미리 예측 가능”

특히, 최근 코로나 바이러스로 인하여 내국인 수요가 국외에서 국내로 집중되면서 제주도 관광지 수요 증가로 이어짐.

이에 따른 제주시 교통량 증가.

대표적인 관광지인 제주시의 교통량을 예측하는 서비스는 여러 방면에서 유용하다고 판단됨.

2. 수집된 데이터_데이콘

데이터 정보 (4가지 유형)_ 날짜/차로 수/도로위치/제한

✓ **데이터 사이즈**

1) Train : 4,701,217
2) Test: 291,241

```
1 print(train.shape , test.shape)
(4701217, 23) (291241, 22)
```

✓ **열 정보**

1) 총 22개 ('id', 'base_date', 'day_of_week', 'base_hour', 'road_in_use', 'lane_count', 'road_rating', 'multi_linked', 'connect_code', 'maximum_speed_limit', 'weight_restricted', 'height_restricted', 'road_type', 'start_latitude', 'start_longitude', 'start_turn_restricted', 'end_latitude', 'end_longitude', 'end_turn_restricted', 'road_name', 'start_node_name', 'end_node_name', 'vehicle_restricted')

2) target 변수 (자동차 평균속도)

✓ **대부분 Categorical 변수**

	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)

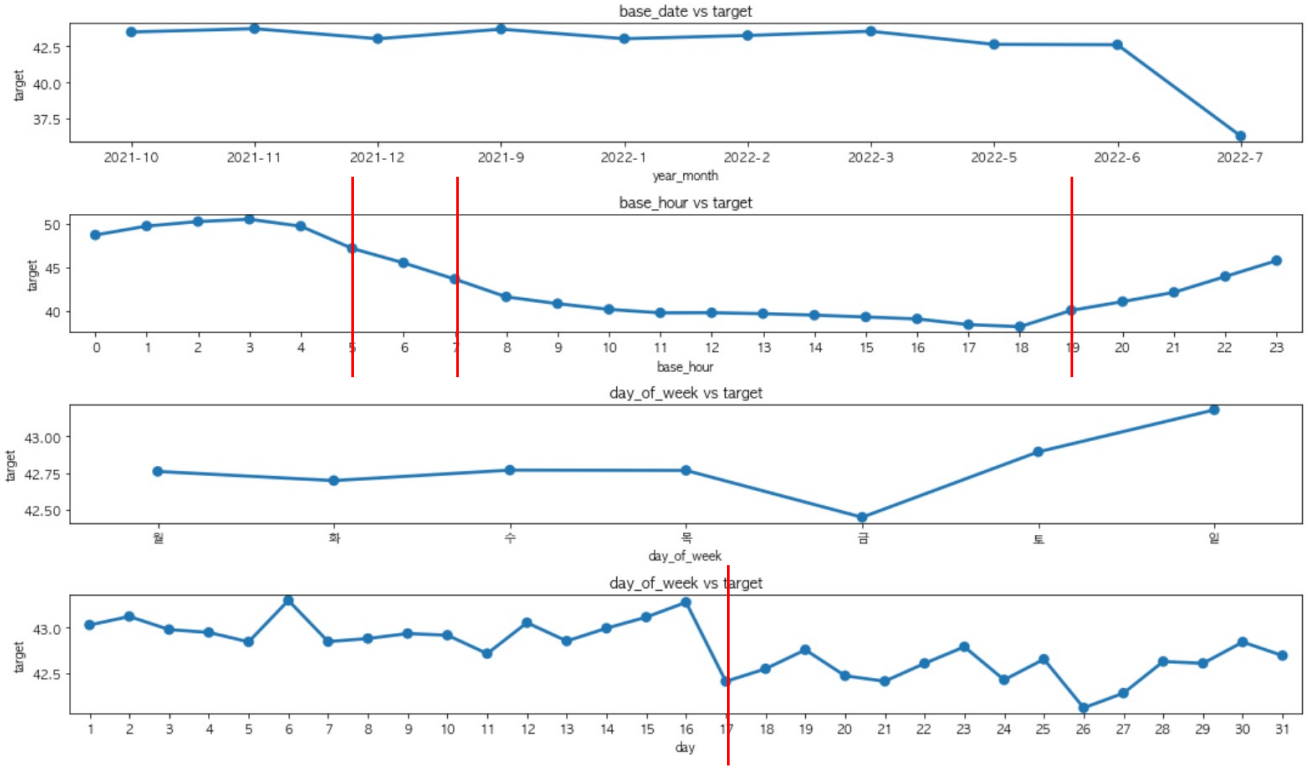
3. 전처리 전략

EDA & Feature Engineering

✓ 날짜 및 시간

- 월별로 차이 별로 없음(또한, test 셋: 2022-8)
- 시간대는 총 4 그룹으로 보여짐
- 요일별로 차이가 별로 없음 (속도차이 최대 0.2)
- 16일 이후, 즉, 매월 중간~말에 통행량 증가

```
#day_s열 만들기 -->16일을 기준으로 2파트로 나누기
train_df.loc[train_df['day']<=16, 'day_s']=0
train_df.loc[train_df['day']>16, 'day_s']=1

# grouping열 만들기
# 그룹1(0시~4시), 그룹2(5시~7시), 그룹3(8시~18시), 그룹4(19시~23시)
g1=[0,1,2,3,4]
g2=[5,6,7]
g3=[8,9,10,11,12,13,14,15,16,17,18]
g4=[19,20,21,22,23]
train_df.loc[train_df['base_hour'].isin(g1), 'grouping']=1
train_df.loc[train_df['base_hour'].isin(g2), 'grouping']=2
train_df.loc[train_df['base_hour'].isin(g3), 'grouping']=3
train_df.loc[train_df['base_hour'].isin(g4), 'grouping']=4
```

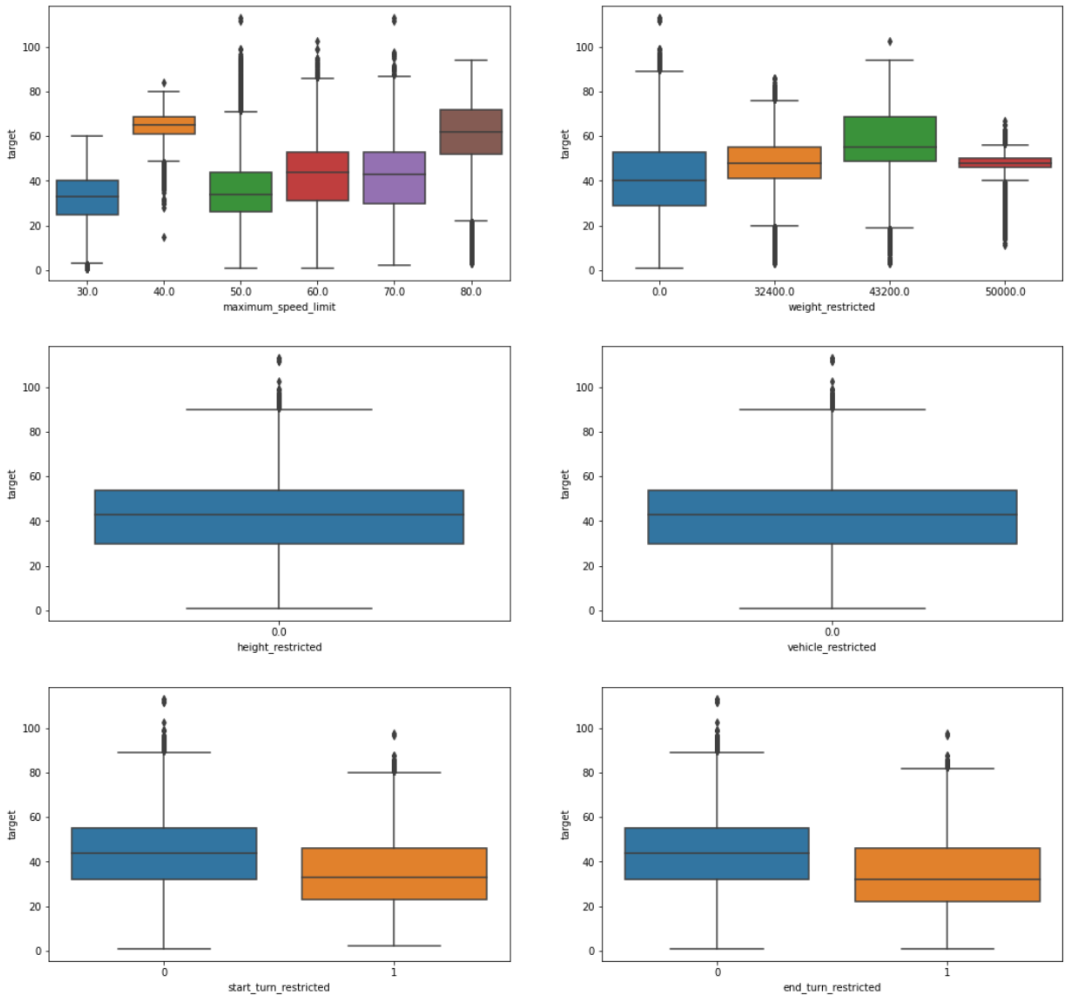


3. 전처리 전략

EDA & Feature Engineering

✓ **제한**

- 최고 속도 제한이 높을수록 평균 속도 증가 예측
  - But, 40에서 평균속도 높음
- Weight 제한이 낮을수록 평균 속도 감소
  - But, 가장 제한이 높을 때 오히려 속도 감소
- Height, vehicle 제한은 값이 0밖에 없음. 훈련에서 제외 결정
- 시작, 도착지점 둘 다 회전 제한이 없을 경우 속도 증가

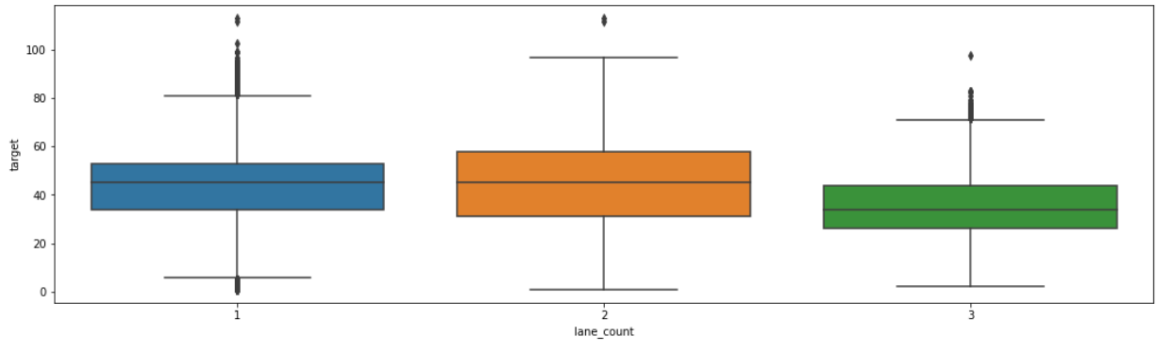


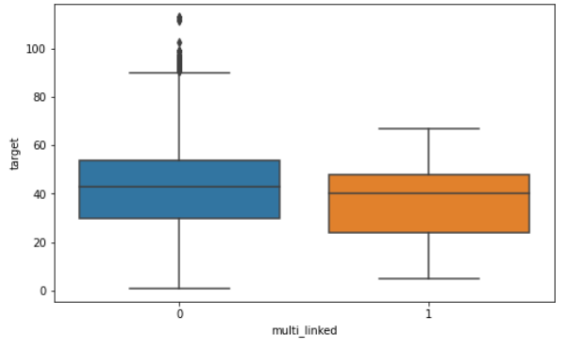
3. 전처리 전략

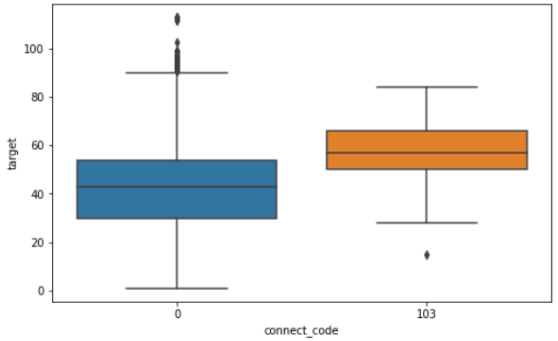
EDA & Feature Engineering

✓ 도로 수

- Lane count가 높을수록 속도 증가 예상
  - But, 오히려 3개일 때 감소
- Multi_linked (중용구간)
  - 존재할 경우 속도가 감소
- Connect_code(연결로)가 있을 때 속도 증가





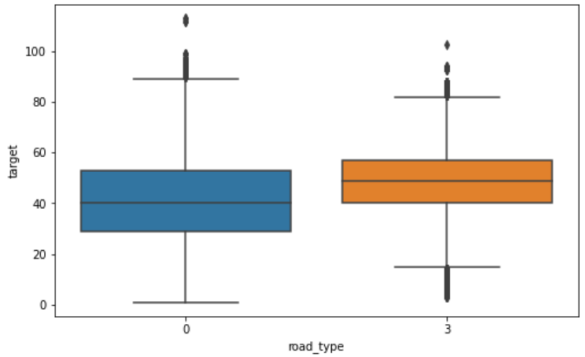


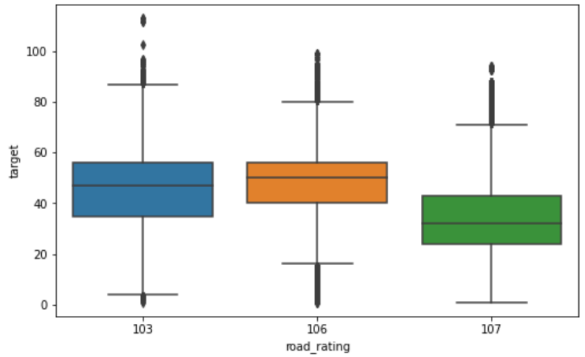
3. 전처리 전략

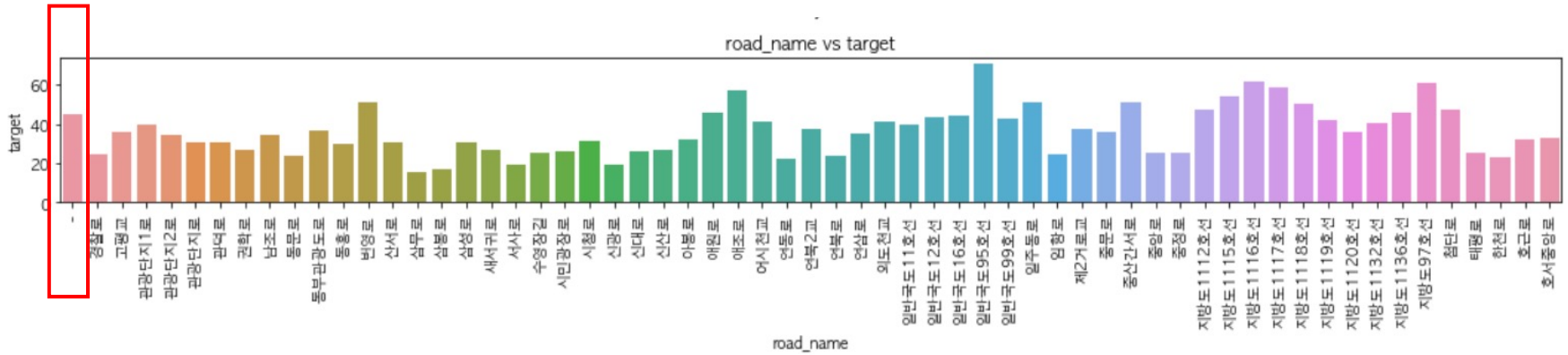
EDA & Feature Engineering

✓ 도로 명

- Road_type 3일때 속도 증가
- Road_rating 107번 도로에서 정체 보임
- Road_name '-' 결측치 존재
- Road name별로 속도 차이 존재 (그룹화 가능해보임)







3. 전처리 전략

EDA & Feature Engineering

✓ 도로명 결측치('-') 변경

- '-' 표시가 있는 도로명 변경 및 삭제

```
# 없어진 것을 확인함
train_df[train_df.road_name=='-']
```

id	base_date	day_of_week	base_hour	lane_count	road_rating	road_name	multi_linked	connect_code

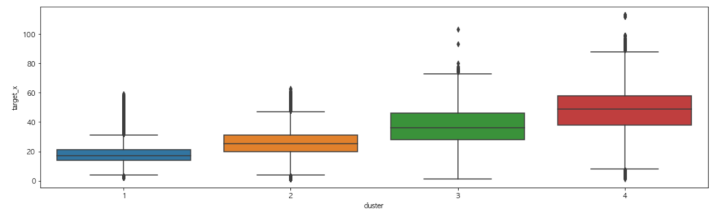
0 rows × 29 columns

```
#107번 도로에서 wighted_restricted={43200:중문로, 32400.0:산서로}
df_107=train_df[train_df.road_rating==107]
print(df_107[df_107['weight_restricted']==43200.0].road_name.value_counts())
print(df_107[df_107['weight_restricted']==0.0].road_name.value_counts())
print(df_107[df_107['weight_restricted']==32400.0].road_name.value_counts())

# 중문로, 산서로 값 채워넣기
train_df1=train_df[train_df.road_name=='-']
train_df1.loc[train_df1['weight_restricted']==43200.0,'road_name']='중문로'
train_df1.loc[train_df1['weight_restricted']==32400.0,'road_name']='산서로'
print(len(train_df1[train_df1.road_name=='-'])) #잘 바뀐것을 확인!

#107번 도로 중 wighted_restricted=0 특징을 알 수 없음..
print(df_107[df_107['weight_restricted']==0.0].describe().iloc[:,11:-1])

# road_name='-'인것 중 weight_restricted=0인거 삭제
jungmoon=train_df1[train_df1.road_name=='중문로']
sanseo=train_df1[train_df1.road_name=='산서로']
train_df=pd.concat([train_df[train_df.road_name!='-'],
                    jungmoon,sanseo],axis=0)
```

✓ 비슷한 속도 별로 도로명 그룹화

```
figure, (ax1) = plt.subplots(nrows=1, ncols=1)
figure.set_size_inches(18,5)

sns.boxplot(data = train_df, x = "cluster", y = "target_x", ax=ax1)
```

```
<AxesSubplot: xlabel='cluster', ylabel='target_x'>
```




```
name1=train_name[train_name.target>=40]
name1['cluster']=4
name2=train_name[(train_name.target<40)&(train_name.target>=30)]
name2['cluster']=3
name3=train_name[(train_name.target<30)&(train_name.target>=20)]
name3['cluster']=2
name4=train_name[(train_name.target<20)]
name4['cluster']=1
all_name=pd.concat([name1,name2,name3,name4], axis=0)

train_df=pd.merge(train_df,all_name, on= 'road_name', how='inner')
train_df.head()
```

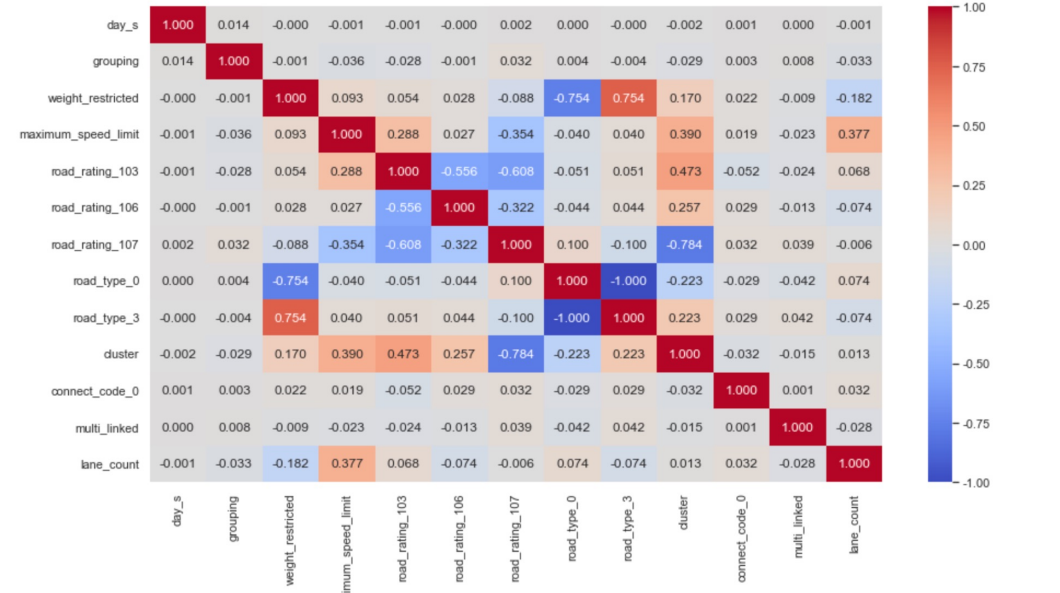
3. 전처리 전략

EDA & Feature Engineering

✓ 상관관계 확인

- ✓ Maximum_speed, weight_restricted는 label encoding
- ✓ Road_type와 weight_restricted 상관관계 높음을 확인
- ✓ Cluster와 road_rating 107과 상관관계 높음
- ✓ Road_type간 상관관계가 높아 road_type 3 남기고 drop

```python
#sns.set(font_scale=0.5)
sns.heatmap(train_df[feature].corr(), annot=True, fmt='.3f', cmap='coolwarm')
sns.set(rc = {'figure.figsize':(15,8)})
```




```python
#day_s열 만들기 -->16일을 기준으로 2파트로 나누기
train_df.loc[train_df['day']<=16, 'day_s']=0
train_df.loc[train_df['day']>16, 'day_s']=1

# grouping열 만들기
# 그룹1(0시~4시), 그룹2(5시~7시), 그룹3(8시~18시), 그룹4(19시~23시)
g1=[0,1,2,3,4]
g2=[5,6,7]
g3=[8,9,10,11,12,13,14,15,16,17,18]
g4=[19,20,21,22,23]
train_df.loc[train_df['base_hour'].isin(g1), 'grouping']=1
train_df.loc[train_df['base_hour'].isin(g2), 'grouping']=2
train_df.loc[train_df['base_hour'].isin(g3), 'grouping']=3
train_df.loc[train_df['base_hour'].isin(g4), 'grouping']=4

train_df=pd.get_dummies(data = train_df, columns = ['road_rating',
                                                    'road_type',
                                                    'start_turn_restricted','connect_coc
                                                    'end_turn_restricted'])
```

```python
# label encoding
cat_features=['maximum_speed_limit','weight_restricted']
for col in cat_features:
    le = LabelEncoder()
    le=le.fit(train_df[col])
    train_df[col]=le.transform(train_df[col])

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

4. 예측 모델_ 적용할 모델 및 하이퍼 파라미터 튜닝

Feature selection

```
feature=['day_s','grouping',
         'weight_restricted','maximum_speed_limit',
         'road_rating_103','road_rating_106','road_rating_107',
         'road_type_3','cluster','connect_code_0',
         'multi_linked','lane_count']
```

+테스트 셋에 존재하지 않는 열 제거
[start_turn_restricted_0', 'start_turn_restricted_1', 'connect_code_103', 'end_turn_restricted_0', 'end_turn_restricted_1']

모델 선언_ 훈련셋, 검증셋 나누기

```
from sklearn.model_selection import train_test_split

#train set
train_df_1=train_df.sample(400000, random_state=37)
y_train = train_df_1['target_x']
X_train = train_df_1[feature]
x_test= test_df[feature]
#validation set
X1_train, X1_val, y1_train, y1_val=train_test_split(X_train, y_train,
                                        test_size=0.2, random_state=156 )
print(X_train.shape,y_train.shape)
print(X1_train.shape)
print(X1_val.shape)
print(x_test.shape)
```

데이터가 너무 많아 400000개 샘플링해서 훈련

```
(400000, 24) (400000,)
(320000, 24)
(80000, 24)
(291241, 24)
```

평가 점수

```
from sklearn.metrics import make_scorer

def rmsle(predict, actual):
    predict=np.array(predict)
    actual =np.array(actual)

    log_predict=np.log(predict+1)
    log_actual=np.log(actual+1)

    difference=log_predict-log_actual
    difference=np.square(difference)

    mean_difference=difference.mean()
    score=np.sqrt(mean_difference)

    return score

rmsle_scorer=make_scorer(rmsle)
rmsle_scorer
```

```
make_scorer(rmsle)
```

4. 예측 모델_ 적용할 모델 및 하이퍼 파라미터 튜닝

RandomForestRegressor

```
hyperparameters_list=[]
n_estimators=1000
num_epoch=15

for epoch in range(num_epoch):
    max_depth=np.random.randint(low=2, high=100)
    max_features=np.random.uniform(low=0.1, high=1.0)

    model=RandomForestRegressor(n_estimators=n_estimators,
                                max_depth=max_depth,
                                max_features=max_features,
                                random_state=37,
                                n_jobs=-1)

    model.fit(X1_train,y1_train)
    y_pred = model.predict(X1_val)

    score=cross_val_score(model, X1_val, y_pred, cv=3, scoring=rmsle_scorer).mean()

    hyperparameters_list.append({'score': score, 'n_estimators': n_estimators,
                                 'max_depth':max_depth, 'max_features':max_features})
    print("Score={0:.5f}".format(score))

hyperparameters_list=pd.DataFrame.from_dict(hyperparameters_list)
hyperparameters_list=hyperparameters_list.sort_values(by='score')
print(hyperparameters_list.shape)
hyperparameters_list.head()
```

	score	n_estimators	max_depth	max_features
13	0.001759	1000	50	0.944471
7	0.001763	1000	24	0.939800
11	0.003074	1000	84	0.730597
4	0.004053	1000	96	0.527461
0	0.004325	1000	81	0.567831

```
#모델
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor(n_estimators=1000,
                            max_depth=50,
                            max_features=0.944471,
                            random_state=27,
                            n_jobs=-1)
model.fit(X_train,y_train)
predictions=model.predict(x_test)
print(predictions.shape)
predictions
```

```
1 list(zip(feature, model.feature_importances_))
```

```
[('day_s', 0.0007663828625872206),
 ('grouping', 0.08107402923054362),
 ('weight_restricted', 0.05320334046297465),
 ('maximum_speed_limit', 0.24324042890850103),
 ('road_rating_103', 0.0497869186204054),
 ('road_rating_106', 0.020791016240332586),
 ('road_rating_107', 0.013056624378113613),
 ('road_type_3', 0.013749964222486656),
 ('cluster', 0.4146272513401184),
 ('connect_code_0', 0.003981403919961971),
 ('multi_linked', 3.823695950780652e-05),
 ('lane_count', 0.1056844028544671)]
```

1. 리더 보드
- 평가 산식 : MAE
- Public score : 전체 테스트 데이터 중 30%
- Private score : 전체 테스트 데이터 중 70%

4. 예측 모델_ 적용할 모델 및 하이퍼 파라미터 튜닝

XGBoostRegressor()

```
hyperparameters_list=[]
n_estimators=1000
num_epoch=11

for epoch in range(num_epoch):
    max_depth=np.random.randint(low=2, high=100)
    learning_rate=np.random.uniform(low=0.1, high=1.0)
    min_child_weight=np.random.uniform(low=1, high=5)
    model=xgb.XGBRegressor(n_estimators=n_estimators,
                           max_depth=max_depth,
                           learning_rate=learning_rate,
                           min_child_weight=min_child_weight,
                           random_state=37,
                           n_jobs=-1)
    model.fit(X1_train,y1_train)
    y_pred = model.predict(X1_val)

    score=cross_val_score(model, X1_val, y_pred,  cv=3, scoring=rmsle_scorer).mean()

    hyperparameters_list.append({'score': score, 'n_estimators': n_estimators,
                                 'max_depth':max_depth, 'learning_rate':learning_rate,
                                'min_child_weight':min_child_weight})
    print("Score={0:.5f}".format(score))

hyperparameters_list=pd.DataFrame.from_dict(hyperparameters_list)
hyperparameters_list=hyperparameters_list.sort_values(by='score')
print(hyperparameters_list.shape)
hyperparameters_list.head()
```

	score	n_estimators	max_depth	learning_rate	min_child_weight
8	0.002267	1000	87	0.775871	2.738758
10	0.002531	1000	34	0.684638	1.311898
1	0.002918	1000	23	0.621695	4.001421
3	0.002981	1000	62	0.609609	3.539444
7	0.003305	1000	14	0.522212	2.738013

```
#모델
model = xgb.XGBRegressor(
        learning_rate = 0.775871,
        max_depth = 87, min_child_weight=2.738758,
        n_estimators = n_estimators,
        objective='reg:linear')

model.fit(X_train, y_train)

predictions=model.predict(x_test)
print(predictions.shape)
predictions
```

```
[('day_s', 0.0006003109441404888),
 ('grouping', 0.08135514511839571),
 ('weight_restricted', 0.08077243677698209),
 ('maximum_speed_limit', 0.26443754782522416),
 ('road_rating_103', 0.03410591761303053),
 ('road_rating_106', 0.03561359061415321),
 ('road_rating_107', 0.09600062329200088),
 ('road_type_3', 0.02550767967695391),
 ('cluster', 0.25477098362889355),
 ('connect_code_0', 0.003985234142876111),
 ('multi_linked', 4.932954841556399e-05),
 ('lane_count', 0.12280120081893382)]
```

5. 예측 모델 개선 전략

- ✓ 교통량은 일반적으로 날씨에 영향을 받음. 따라서 날씨 데이터를 추가할 수 있음
- ✓ 도로 주변 명소 방문객 수치 데이터를 추가할 수 있음
- ✓ 특성 공학을 통해 또 다른 특성 생성 가능 (특히 상관관계가 높았던 열)
- ✓ 다양한 하이퍼 파라미터 조합을 시도해 봄으로써 성능을 향상할 수 있음
- ✓ 다른 모델을 시도할 수 있음

6. 활용 전략

제주도 교통량 예측 분석을 통해 교통지원정책 및 비즈니스 활용

✓ **제주도 관광지의 높은 수요와 이에 따른 교통량 증가 문제 해결 가능**

- 관광지 특성상 관광시즌의 영향을 많이 받음
- 지역 자체의 교통량 특성이 존재하기 때문에 분석을 하는 것은 여러 면에서 이점 존재

✓ **교통지원정책 활용 방안**

- 교통량 분석을 통해 적절한 시점, 위치에 대중교통 수단을 적절히 배치 가능
- 교통량에 따라 신호등을 자동으로 변경하는 스마트 교통 관리 시스템을 위해 활용 가능 (스마트 시티)

✓ **비즈니스 활용 가능**

- 교통량이 많은 지역을 탐지하여 상권 분석에 활용 가능
- 교통 예보 서비스 제공 가능 등.

감사합니다 ☺