

퇴근시간 버스승차인원 예측

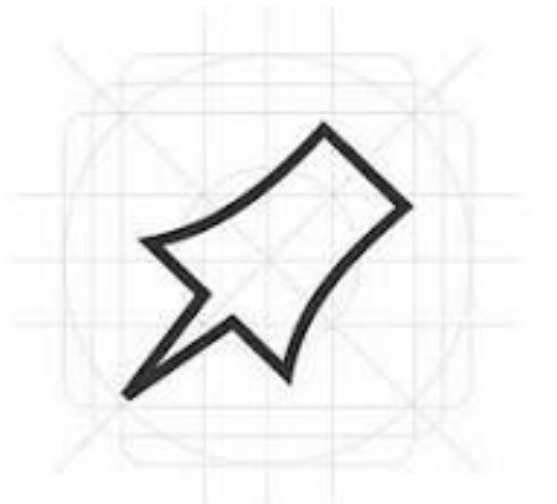
- 유원희
- 황성현
- 김도현





목차

1. 주제 선정 배경
2. 목표
3. 데이터 수집 및 출처
4. 전처리
5. 모델링
6. 평가





주제 선정 배경



제주도내 주민등록인구는 2019년 11월 기준 69만명으로, 연평균 4%대로 성장했습니다.

외국인과 관광객까지 고려하면 전체 상주인구는 90만명을 넘을 것으로 추정됩니다.

제주도민 증가와 외국인의 증가로 현재 제주도의 교통체증이 심각한 문제로 떠오르고 있습니다.

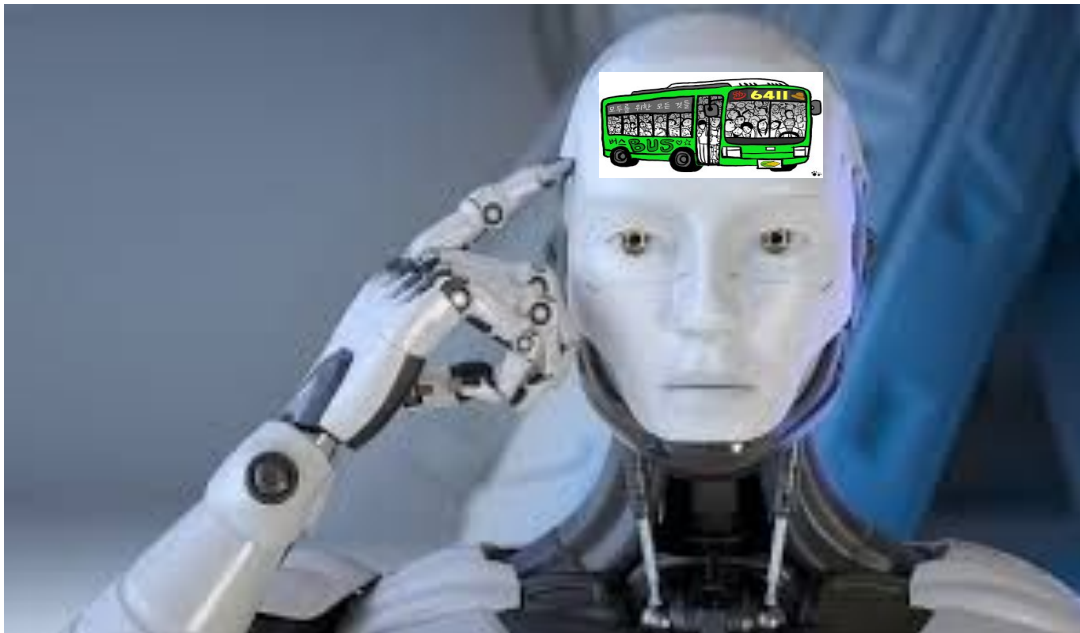
2017년 한국은행 제주본부에 따르면 제주도 일부 지역은 교통체증이 서울보다 심각합니다.

따라서, 제주테크노파크는 교통난 심화를 해결하기 위해 데이터 분석 대회를 개최합니다.



목표

제주도 버스의 효율적인 운영을 위해 퇴근시간 승차인원을 예측하는 모델 개발





데이터 수집 및 출처



▶ 기존 대회 데이터(dacon 대회 데이터) : train.csv , test.csv, bus_bts.csv

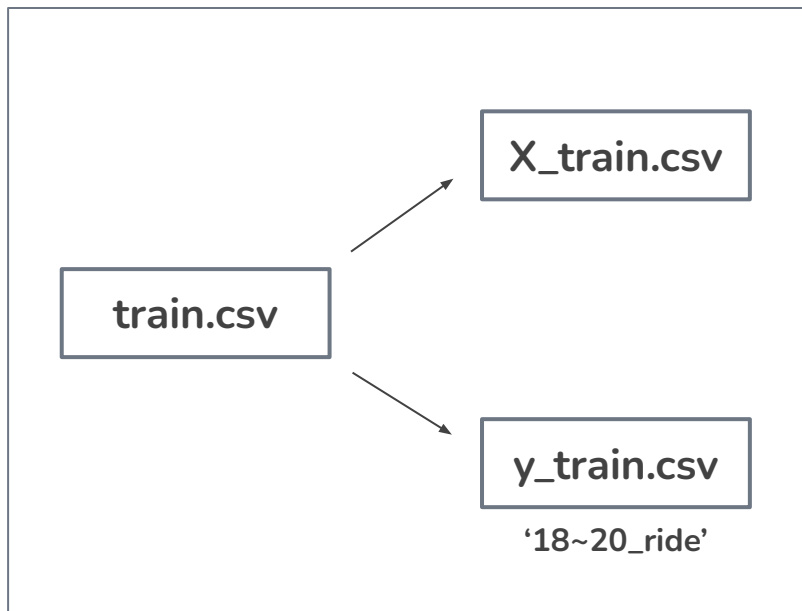
출처 : <https://dacon.io/competitions/official/229255/data>

▶ 외부 데이터 : 2019년도 9~10월 기상정보 (기상자료개방포털)

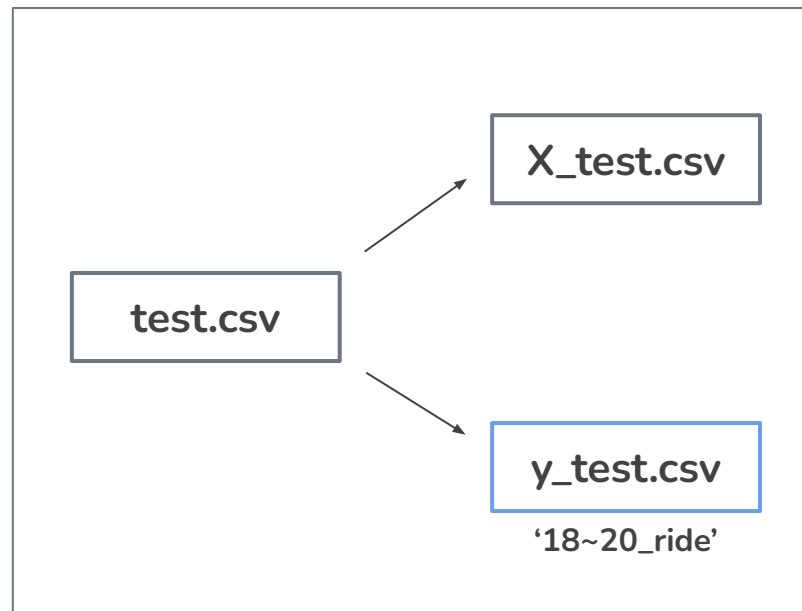
출처 : <https://data.kma.go.kr/cmmn/main.do>)



Feature, Target 분리



9/1 ~ 9/30



10/1 ~ 10/16



전처리

□ 파생 변수 생성

1. $\text{route_station} = \text{bus_route_id} + \text{station_code}$
 - 특정 노선-정류소 조합에 대한 패턴을 더 쉽게 분석할 수 있다.(중요)
2. $\text{bus_route_id_weekday} = \text{bus_route_id} + \text{weekday}$
 - 특정 요일에 대한 승하차 패턴을 분석
3. $\text{station_code_weekday} = \text{station_code} + \text{weekday}$
 - 특정 정류소의 특정 요일에 발생하는 승하차 데이터를 고유하게 식별
4. $\text{route_station_weekday} = \text{route_station} + \text{weekday}$
 - 특정 노선의 특정 정류소에서 특정 요일에 발생하는 패턴을 더 세밀하게 분석
5. on_time
 - target이 18~20 으로 2시간 이므로 , 승 하차 시간대 통합 변수 생성 ($t \sim t+2$)



전처리

bus_route_id + station_code => route_station

	bus_route_id	station_code
0	4270000	344
1	4270000	357
2	4270000	432
3	4270000	1579
4	4270000	1646
...
415418	32820000	1129
415419	32820000	1564
415420	32820000	2322
415421	32820000	3291
415422	32820000	6115100



	route_station
0	4270000,344
1	4270000,357
2	4270000,432
3	4270000,1579
4	4270000,1646
...	...
415418	32820000,1129
415419	32820000,1564
415420	32820000,2322
415421	32820000,3291
415422	32820000,6115100



전처리

bus_route_id + weekday => bus_route_id_weekday

	bus_route_id	weekday
0	4270000	6
1	4270000	6
2	4270000	6
3	4270000	6
4	4270000	6
...
415418	32820000	0
415419	32820000	0
415420	32820000	0
415421	32820000	0
415422	32820000	0



	bus_route_id_weekday
0	4270000,6
1	4270000,6
2	4270000,6
3	4270000,6
4	4270000,6
...	...
415418	32820000,0
415419	32820000,0
415420	32820000,0
415421	32820000,0
415422	32820000,0



전처리

station_code + weekday => station_code_weekday

	station_code	weekday
0	344	6
1	357	6
2	432	6
3	1579	6
4	1646	6
...
415418	1129	0
415419	1564	0
415420	2322	0
415421	3291	0
415422	6115100	0



	station_code_weekday
0	344,6
1	357,6
2	432,6
3	1579,6
4	1646,6
...	...
415418	1129,0
415419	1564,0
415420	2322,0
415421	3291,0
415422	6115100,0



전처리

route_station + weekday => route_station_weekday

	route_station	weekday
0	4270000,344	6
1	4270000,357	6
2	4270000,432	6
3	4270000,1579	6
4	4270000,1646	6
...
415418	32820000,1129	0
415419	32820000,1564	0
415420	32820000,2322	0
415421	32820000,3291	0
415422	32820000,6115100	0



	route_station_weekday
0	4270000,344,6
1	4270000,357,6
2	4270000,432,6
3	4270000,1579,6
4	4270000,1646,6
...	...
415418	32820000,1129,0
415419	32820000,1564,0
415420	32820000,2322,0
415421	32820000,3291,0
415422	32820000,6115100,0



전처리

bts.csv 의 'geton_time' 컬럼 시간대만 추출 => on_time

geton_time
06:34:45
06:34:58
07:19:07
09:14:47
09:28:53
...
07:08:31
07:16:31
08:29:05
08:40:32
08:40:35

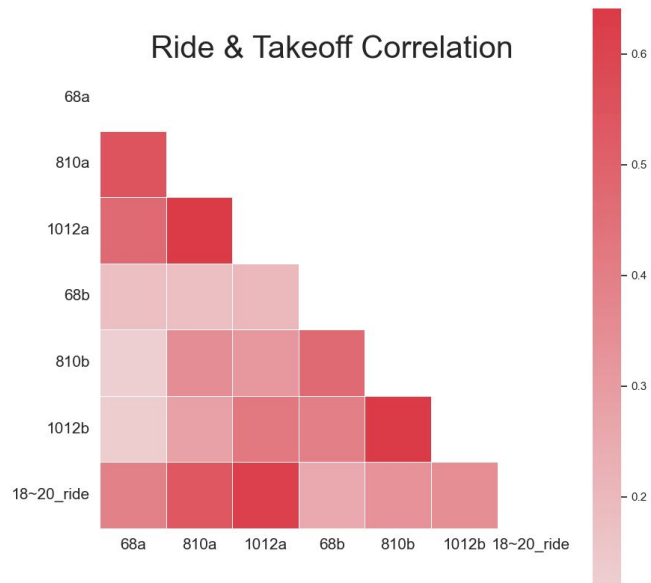


on_time
6~7_ride
6~7_ride
7~8_ride
9~10_ride
9~10_ride
...
7~8_ride
7~8_ride
8~9_ride
8~9_ride
8~9_ride



전처리

EDA

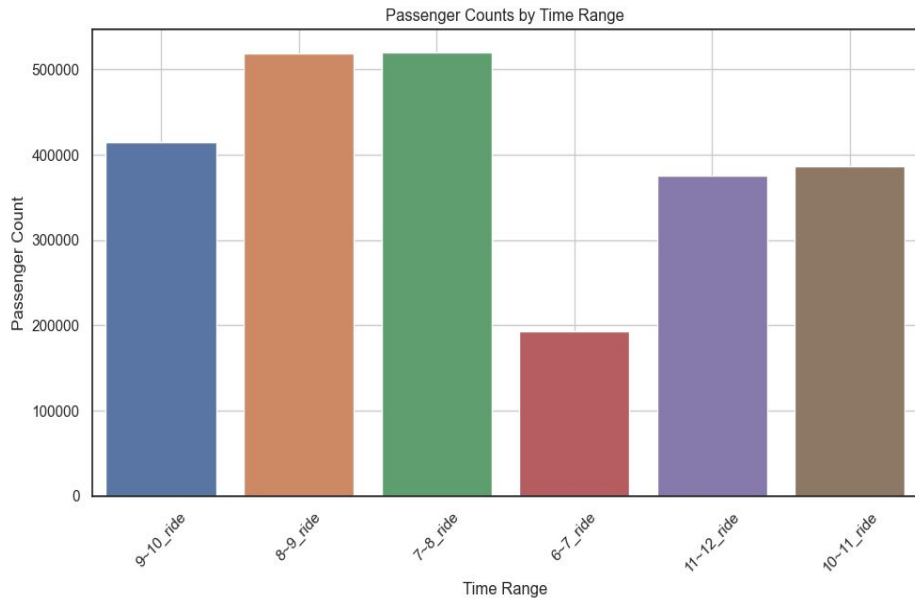


오전 8-10시, 10-12시의 승차인원이 18-20_ride와 높은 상관관계를 보임



전처리

EDA



12시 이전에는 7시부터9시 사이의 이용객이 많은 것을 확인



전처리

EDA 후 파생 변수 생성

6~7_ride	7~8_ride	8~9_ride	9~10_ride	10~11_ride	11~12_ride
0.0	1.0	2.0	5.0	2.0	6.0
1.0	4.0	4.0	2.0	5.0	6.0
1.0	1.0	0.0	2.0	0.0	0.0
0.0	17.0	6.0	26.0	14.0	16.0
0.0	0.0	0.0	0.0	0.0	0.0
...
4.0	0.0	0.0	0.0	0.0	0.0
4.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0



68a	810a	1012a
1.0	7.0	8.0
5.0	6.0	11.0
2.0	2.0	0.0
17.0	32.0	30.0
0.0	0.0	0.0
...
4.0	0.0	0.0
4.0	0.0	0.0
0.0	0.0	0.0
1.0	0.0	0.0
0.0	0.0	0.0

승 하차 시간대 2시간 간격 변수 생성



전처리

최종적으로 예측해야할 Target

- 각 일자별 (date),
- 버스노선 (bus_route_id)상의
- 정류장 (station_name)의
- 퇴근시간 하차인원 (18~20_ride)



전처리

```
train, test = id_statistic('route_station', '1820_rs_mean', '1820_rs_sum')  
  
train.shape, test.shape
```

```
train, test = id_statistic('bus_route_id', '1820_r_mean', '1820_r_sum')  
  
train.shape, test.shape
```

```
train, test = id_statistic('station_code', '1820_s_mean', '1820_s_sum')  
  
train.shape, test.shape
```

```
train, test = id_statistic('weekday', '1820_w_mean', '1820_w_sum')  
  
train.shape, test.shape
```



1820_w_mean	1820_w_sum	1820_s_mean	1820_s_sum	1820_r_mean	1820_r_sum	1820_rs_mean	1820_rs_sum
1.034282	54306.0	1.466667	44.0	3.104381	2409.0	1.466667	44.0
1.034282	54306.0	4.178218	6330.0	3.104381	2409.0	5.366667	161.0
1.034282	54306.0	2.169559	1625.0	3.104381	2409.0	1.600000	48.0
1.034282	54306.0	52.032258	1613.0	3.104381	2409.0	53.766667	1613.0
1.034282	54306.0	0.732794	181.0	3.104381	2409.0	0.153846	4.0
...
1.343710	104073.0	2.393089	1108.0	0.000000	0.0	0.000000	0.0
1.343710	104073.0	7.726008	5555.0	0.000000	0.0	0.000000	0.0
1.343710	104073.0	0.320652	59.0	0.000000	0.0	0.000000	0.0
1.343710	104073.0	1.333333	160.0	0.000000	0.0	0.000000	0.0
1.343710	104073.0	0.188636	83.0	0.000000	0.0	0.000000	0.0

각 컬럼(route_station, bus_route_id, station_code, weekday)별 18~20_ride(평균 값, 더한 값)의 파생 변수 생성



전처리

bus_route_id => congestion(18~20_ride 혼잡도)

```
def congestion():
    # 버스 노선별 총 승객 수 계산
    df = train.groupby(['bus_route_id'])['18~20_ride'].agg(['passenger', 'sum']).reset_index()

    # 승객 수에 따른 혼잡도 레벨 지정
    def get_congestion_level(passenger):
        if passenger > 10000:
            return 7
        elif passenger > 5000:
            return 6
        elif passenger > 2000:
            return 5
        elif passenger > 700:
            return 4
        elif passenger > 200:
            return 3
        elif passenger > 50:
            return 2
        else:
            return 1

    df['congestion'] = df['passenger'].apply(get_congestion_level)
    df = df[['bus_route_id', 'congestion']]

    # 학습 데이터와 테스트 데이터에 혼잡도 정보 병합
    tr = pd.merge(train, df, how='left', on='bus_route_id')
    te = pd.merge(test, df, how='left', on='bus_route_id')

    # 테스트 데이터의 결측치 처리(중앙값으로)
    te['congestion'] = te['congestion'].fillna(df['congestion'].median())

    return tr, te
```



congestion	
	5
	5
	5
	5
	5
	...
	1
	1
	1
	1
	1



전처리

```
bts['geton_time2'] = pd.to_datetime(bts['geton_time'])

f = bts.groupby(['geton_date', 'geton_time2', 'geton_station_code', 'bus_route_id'])['user_count'].\\
agg(['탑승객_수', 'sum']).reset_index().\\
sort_values(by=['geton_date', 'geton_station_code', 'bus_route_id', 'geton_time2'], ascending=True).reset_index()

f['index'] = list(range(0, len(f)))
```

```
time = []

for i in range(0, len(f)-1): # 처음부터 마지막 인덱스 전까지

    if ((f.iloc[i].geton_date == f.iloc[i+1].geton_date) &\\
        (f.iloc[i].geton_station_code == f.iloc[i+1].geton_station_code) &\\
        (f.iloc[i].bus_route_id == f.iloc[i+1].bus_route_id)):

        time.append(f.iloc[i+1].geton_time2 - f.iloc[i].geton_time2)

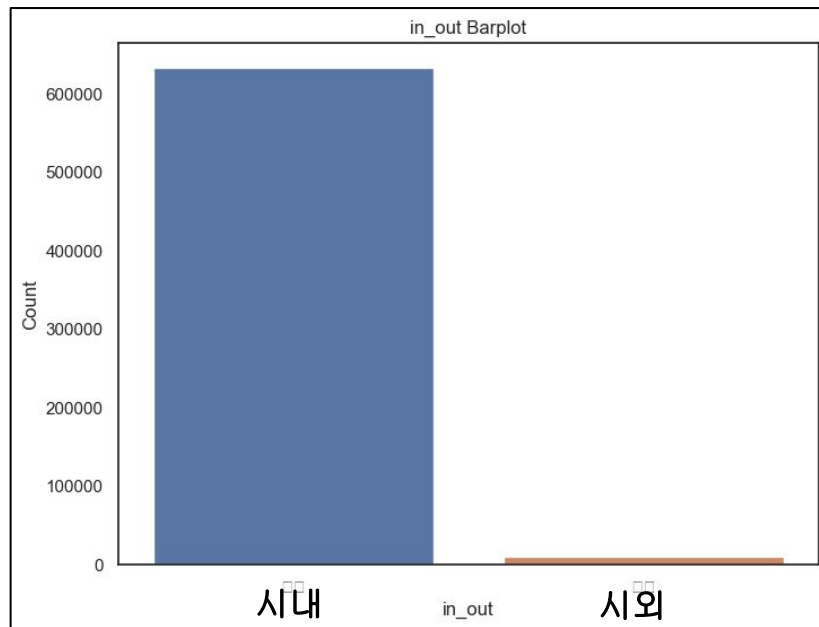
    else:
        time.append(0)

time.insert(0, '0') # 앞에 시간을 구분하기 위해 0표시
```

배차 간격도 예측하는데 관련이 있을 거 같아 변수로 사용



전처리

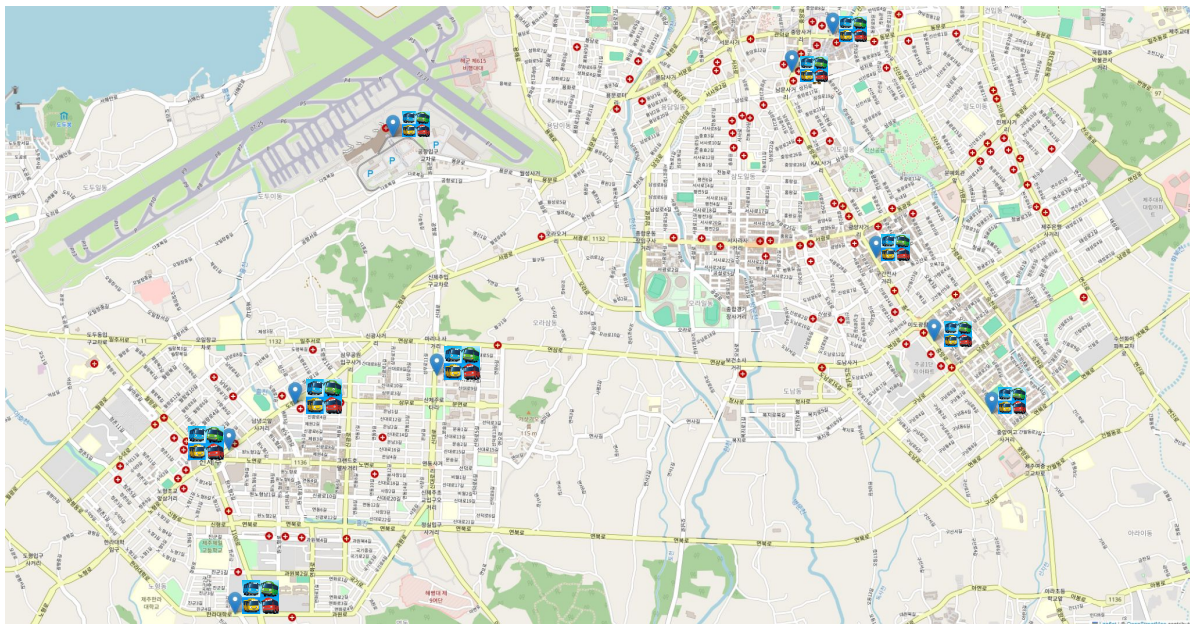


시내와 시외를 0 과 1로 변환



전처리

거리에 따른 파생변수 생성



제주시와 주변 주요 지점과의 거리를 계산함으로써, 특정 버스 정류장이 얼마나 접근성이 좋은지 평가



전처리

제주시와 주변 주요 지점과의 거리를 계산하여 변수 생성

```
coords_jeju1 = (33.500770, 126.522761) #제주시의 위도 경도
data['dis_jeju1'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_jeju1).km for i in range(len(data))]

coords_jejusicheong1 = (33.49892, 126.53035) #제주시청(광안방면)의 위도 경도
coords_jejuairport = (33.50661, 126.49345) #제주국제공항(구제주방면)의 위도 경도
coords_hallahosp = (33.48963, 126.486) #한라병원의 위도 경도
coords_rotary = (33.49143, 126.49678) # 제주시청제주도청의 위도 경도
coords_jejucenterhigh = (33.48902, 126.5392) #제주중앙여자고등학교의 위도 경도
coords_jejumarket = (33.51315, 126.52786) #중문시장의 위도 경도
coords_jejusclass = (33.47626, 126.48141) #제주고등학교/중문5출입구의 위도 경도
coords_centerroad = (33.51073, 126.5239) #중문로(국민은행)의 위도 경도
coords_fiveway = (33.48667, 126.48092) # 노원오거리의 위도 경도
coords_law = (33.49363, 126.53476) # 제주시법원(광안방면)의 위도 경도

data['dis_jejusicheong1'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_jejusicheong1).km for i in range(len(data))]
data['dis_jejuairport'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_jejuairport).km for i in range(len(data))]
data['dis_hallahosp'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_hallahosp).km for i in range(len(data))]
data['dis_rotary'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_rotary).km for i in range(len(data))]
data['dis_jejucenterhigh'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_jejucenterhigh).km for i in range(len(data))]
data['dis_jejumarket'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_jejumarket).km for i in range(len(data))]
data['dis_jejusclass'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_jejusclass).km for i in range(len(data))]
data['dis_centerroad'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_centerroad).km for i in range(len(data))]
data['dis_fiveway'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_fiveway).km for i in range(len(data))]
data['dis_law'] = [geodesic((data['latitude'].iloc[i], data['longitude'].iloc[i]), coords_law).km for i in range(len(data))]

data.shape
```



전처리

출근 시간별 총 승객

```
data['ride_sum'] = data['6~7_ride'] + data['7~8_ride'] + data['8~9_ride'] + data['9~10_ride'] + data['10~11_ride'] + data['11~12_ride']  
data['takeoff_sum'] = data['6~7_takeoff'] + data['7~8_takeoff'] + data['8~9_takeoff'] + data['9~10_takeoff'] + data['10~11_takeoff'] + data['11~12_takeoff']
```

날짜 및 시간대별 총 승객

```
f = data.groupby('date')['6~7_ride'].agg(['6~7_all_ride_number', 'sum']).reset_index()  
data = pd.merge(data, f, how='left')  
  
f = data.groupby('date')['7~8_ride'].agg(['7~8_all_ride_number', 'sum']).reset_index()  
data = pd.merge(data, f, how='left')  
  
f = data.groupby('date')['8~9_ride'].agg(['8~9_all_ride_number', 'sum']).reset_index()  
data = pd.merge(data, f, how='left')  
  
f = data.groupby('date')['9~10_ride'].agg(['9~10_all_ride_number', 'sum']).reset_index()  
data = pd.merge(data, f, how='left')  
  
f = data.groupby('date')['10~11_ride'].agg(['10~11_all_ride_number', 'sum']).reset_index()  
data = pd.merge(data, f, how='left')
```




전처리

요일 별 Target 평균 수 생성

```
def week_mean(data):  
    # 데이터프레임 초기화  
    df = data.reset_index(drop=True)  
  
    # 요일별 평균값 계산  
    weekday_means = df.groupby('weekday')['18~20_ride'].mean()  
  
    # 새로운 열 'weekdaymean' 추가 및 기본값 설정  
    df['weekdaymean'] = 1  
  
    # 요일별 평균값 할당  
    for weekday, mean_value in weekday_means.items():  
        df.loc[df['weekday'] == weekday, 'weekdaymean'] = mean_value  
  
    return df
```

	weekdaymean
0	1.034282
1	1.034282
2	1.034282
3	1.034282
4	1.034282
...	...
643588	1.430856
643589	1.430856
643590	1.430856
643591	1.430856
643592	1.430856



전처리

각각의 측정소와 정류장 사이의 거리를 계산한 파생 변수 생성

```
def dist() :  
    jeju=(33.51411, 126.52969) # 제주 측정소 근처  
    gosan=(33.29382, 126.16283) #고산 측정소 근처  
    seongsan=(33.38677, 126.8802) #성산 측정소 근처  
    po=(33.24616, 126.5653) #서귀포 측정소 근처  
  
    t1 = [ (import) geodesic: Any for i,j in list( zip( data['latitude'],data['longitude'] )) ]  
    t2 = [geodesic( (i,j), gosan).km for i,j in list( zip( data['latitude'],data['longitude'] )) ]  
    t3 = [geodesic( (i,j), seongsan).km for i,j in list( zip( data['latitude'],data['longitude'] )) ]  
    t4 = [geodesic( (i,j), po).km for i,j in list( zip( data['latitude'],data['longitude'] )) ]  
  
    data['dis_jeju'] = t1  
    data['dis_gosan']=t2  
    data['dis_seongsan']=t3  
    data['dis_po']=t4  
  
    total = pd.DataFrame(list(zip( t1,t2,t3,t4)),columns=['jeju','gosan','seongsan','po'] )  
    data['dist_name'] = total.apply(lambda x: x.argmin(), axis=1)  
  
    return data
```



전처리

비 오는 날과, 비 안오는 날 0과 1로 변환

```
# rainy_day
# 비 오는날=1, 비 안오는 날=0
def f(x):
    if x == 0:
        return 0
    else:
        return 1
```



1차 모델링

2개의 X 변수 데이터셋 생성

```
input_var1 = ['in_out', 'latitude', 'longitude', '6~7_ride', '7~8_ride', '8~9_ride', '9~10_ride', '10~11_ride', '11~12_ride',  
'6~7_takeoff', '7~8_takeoff', '8~9_takeoff', '9~10_takeoff', '10~11_takeoff', '11~12_takeoff',  
'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4', 'weekday_5', 'weekday_6',  
'dis_jeju1', 'dis_jejusicheong1', 'dis_jejuairport', 'dis_hallahosp', 'dis_rotary', 'dis_jejucenterhigh',  
'dis_jejuemarket', 'dis_centerroad', 'dis_jejuclass', 'dis_fiveway', 'dis_law',  
'weekend', 'holiday', 'ride_sum', 'takeoff_sum', '1820_rs_mean', '1820_r_mean', '1820_s_mean', 'congestion',  
'station_code2', 'bus_route_id2', '일강수_10', '현재일기_10', '평강온도_10',  
'6~7_all_ride_number', '7~8_all_ride_number', '8~9_all_ride_number', '9~10_all_ride_number', '10~11_all_ride_number',  
'1820_w_mean', 'in_out_mean', 'weekdaymean', 'adult', 'kids', 'teen', 'elder', 'adult_prop', 'kids_prop', 'teen_prop', 'elder_prop',  
'mean_job_majorc', 'mean_job_smallc', 'mean_job_public', 'mean_job_profession', 'mean_job_self',  
'mean_vehicle_own_rat', 'mean_avg_income', 'mean_med_income', 'mean_avg_spend',  
'rate_job_majorc', 'rate_job_smallc', 'rate_job_public', 'rate_job_profession', 'rate_job_self',  
'rate_vehicle_own_rat', 'rate_avg_income', 'rate_med_income', 'rate_avg_spend',  
'sum_job_majorc', 'sum_job_smallc', 'sum_job_public', 'sum_job_profession', 'sum_job_self',  
'sum_vehicle_own_rat', 'sum_avg_income', 'sum_med_income', 'sum_avg_spend',  
'68a', '810a', '1012a', '68b', '810b', '1012b', '69a', '912a', '69b', '912b',  
'dis_jeju', 'dis_gosan', 'dis_seongsan', 'dis_po', '기온(°C)', '강수량(mm)',  
'dist_name_gosan', 'dist_name_jeju', 'dist_name_po', 'dist_name_seongsan', 'si_서귀포시', 'si_제주시',  
'school', 'transfer', 'dong2', 'rainy_day']
```

var1

```
input_var3 = ['sum_avg_spend', '68a', '810a', 'si_제주시', 'dong2', 'bus_interval', 'dis_jejuairport', 'ride_sum',  
'takeoff_sum', '1820_rs_mean', '1820_rs_sum', '1820_r_mean', '1820_r_sum', '1820_s_mean',  
'1820_s_sum', 'congestion', 'bus_route_id2', '6~7_all_ride_number', '7~8_all_ride_number',  
'8~9_all_ride_number', '1012a_mean', '1012b_sum', '10~11_ride_sum', '10~11_takeoff_sum', '11~12_ride_sum',  
'11~12_takeoff_sum', '1820_r_mean_sum', '1820_r_mean_mean',  
'1820_r_sum_sum', '1820_r_sum_mean', '1820_rs_mean_sum',  
'1820_s_mean_sum', '1820_s_mean_mean', '1820_s_sum_sum',  
'1820_s_sum_mean', '1820_w_mean_sum', '1820_w_mean_mean',  
'1820_w_sum_mean', '68a_sum', '68a_mean', '68b_sum', 'in_out', 'latitude', 'longitude',  
'6~7_ride', '7~8_ride', '8~9_ride', '9~10_ride', '10~11_ride', '11~12_ride', '6~7_takeoff', '7~8_takeoff',  
'8~9_takeoff', '9~10_takeoff', '10~11_takeoff', '11~12_takeoff',  
'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4',  
'weekday_5', 'weekday_6', 'dis_jeju1', '68b_mean', '6~7_ride_sum',  
'6~7_ride_mean', '6~7_takeoff_sum', '6~7_takeoff_mean', '7~8_ride_sum',  
'7~8_ride_mean', '7~8_takeoff_sum', '7~8_takeoff_mean', '810a_sum',  
'810b_sum', '8~9_ride_sum', '8~9_takeoff_sum', '8~9_takeoff_mean',  
'9~10_ride_sum', '9~10_takeoff_sum', 'route_station_weekday2']
```

var2



1차 모델링

var1으로 LightGBM 학습 (GridSearchCV)

GridSearch로 하이퍼파라미터최적화 (var1)

```
import lightgbm as lgb
lgbm = lgb.LGBMRegressor()

param_grid = {
    'n_estimators': [500],
    'num_leaves': [31, 50],
    'learning_rate': [0.05],
    'min_child_samples': [20, 50],
    'max_depth': [-1, 10],
    'feature_fraction': [0.8, 1.0],
    'bagging_fraction': [0.8, 1.0],
    'lambda_l1': [0, 0.1],
    'lambda_l2': [0, 0.1],
    'boosting_type': ['dart'],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator = lgbm, param_grid = param_grid, cv = 3, scoring = 'neg_mean_squared_error',
                           verbose = 1, n_jobs = -1)

grid_search.fit(X_train1[my_var1], y_train)

print("Best parameters : ", grid_search.best_params_)

best_model = grid_search.best_estimator_
X_test['18~20_ride'] = best_model.predict(X_test[my_var1])
X_test[['id', '18~20_ride']].to_csv("lightgbm_gridCV_result.csv", index=False)
```



1차 모델링

var1으로 LightGBM 학습 (GridSearchCV)

GridSearch로 하이퍼파라미터최적화 (var1)

```
import lightgbm as lgb
lgbm = lgb.LGBMRegressor()

param_grid = {
    'n_estimators': [500],
    'num_leaves': [31, 50],
    'learning_rate': [0.05],
    'min_child_samples': [20, 50],
    'max_depth': [-1, 10],
    'feature_fraction': [0.8, 1.0],
    'bagging_fraction': [0.8, 1.0],
    'lambda_l1': [0, 0.1],
    'lambda_l2': [0, 0.1],
    'boosting_type': ['dart'],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator = lgbm, param_grid = param_grid, cv = 3, scoring = 'neg_mean_squared_error',
                           verbose = 1, n_jobs = -1)

grid_search.fit(X_train1[my_var1], y_train)

print("Best parameters : ", grid_search.best_params_)

best_model = grid_search.best_estimator_
X_test[['18~20_ride']] = best_model.predict(X_test[my_var1])
X_test[['id', '18~20_ride']].to_csv("lightgbm_gridCV_result.csv", index=False)
```

➡ RMSE : 2.2461403605



1차 모델링

var2으로 LightGBM 학습 (GridSearchCV)

```
import lightgbm as lgb
lgbm = lgb.LGBMRegressor()

param_grid = {
    'n_estimators': [500],
    'num_leaves': [31],
    'learning_rate': [0.05],
    'min_child_samples': [20],
    'max_depth': [10],
    'feature_fraction': [0.8],
    'bagging_fraction': [0.8],
    'lambda_l1': [0],
    'lambda_l2': [0.1],
    'boosting_type': ['dart'],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}

grid_search = GridSearchCV(estimator = lgbm, param_grid = param_grid, cv = 3, scoring = 'neg_mean_squared_error',
                           verbose = 1, n_jobs = -1)

grid_search.fit(X_train[my_var3], y_train)

print("Best parameters : ", grid_search.best_params_)

best_model = grid_search.best_estimator_
X_test['18~20_ride'] = best_model.predict(X_test[my_var3])
X_test[['id', '18~20_ride']].to_csv("lightgbm_gridCV_result_var3.csv", index=False)
```



1차 모델링

var2으로 LightGBM 학습 (GridSearchCV)

```
import lightgbm as lgb
lgbm = lgb.LGBMRegressor()

param_grid = {
    'n_estimators': [500],
    'num_leaves': [31],
    'learning_rate': [0.05],
    'min_child_samples': [20],
    'max_depth': [10],
    'feature_fraction': [0.8],
    'bagging_fraction': [0.8],
    'lambda_l1': [0],
    'lambda_l2': [0.1],
    'boosting_type': ['dart'],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}

grid_search = GridSearchCV(estimator = lgbm, param_grid = param_grid, cv = 3, scoring = 'neg_mean_squared_error',
                           verbose = 1, n_jobs = -1)

grid_search.fit(X_train[my_var3], y_train)

print("Best parameters : ", grid_search.best_params_)

best_model = grid_search.best_estimator_
X_test['18~20_ride'] = best_model.predict(X_test[my_var3])
X_test[['id', '18~20_ride']].to_csv("lightgbm_gridCV_result_var3.csv", index=False)
```

➡ RMSE : 2.2268666474



1차 모델링

var2으로 LightGBM 학습 (GridSearchCV)

```
import lightgbm as lgb
lgbm = lgb.LGBMRegressor()

param_grid = {
    'n_estimators': [500],
    'num_leaves': [31],
    'learning_rate': [0.05],
    'min_child_samples': [20],
    'max_depth': [10],
    'feature_fraction': [0.8],
    'bagging_fraction': [0.8],
    'lambda_l1': [0],
    'lambda_l2': [0.1],
    'boosting_type': ['dart'],
    'subsample': [0.8],
    'colsample_bytree': [0.8]
}

grid_search = GridSearchCV(estimator = lgbm, param_grid = param_grid, cv = 3, scoring = 'neg_mean_squared_error',
                           verbose = 1, n_jobs = -1)

grid_search.fit(X_train[my_var3], y_train)

print("Best parameters : ", grid_search.best_params_)

best_model = grid_search.best_estimator_
X_test['18~20_ride'] = best_model.predict(X_test[my_var3])
X_test[['id', '18~20_ride']].to_csv("lightgbm_gridCV_result_var3.csv", index=False)
```

➡ RMSE : 2.2268666474



1차 모델링 최저 RMSE



2차 모델링

LightGBM으로 var1에 대해 Feature Selection

LightGBM 의 feature selection

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer

# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X_train[my_var1], y_train, test_size=0.3, random_state=123)

model = lgb.LGBMRegressor()
model.fit(X_train, y_train)

# Feature importance 계산
importance = model.feature_importances_
feature_importance = pd.DataFrame({'feature': my_var1, 'importance': importance})
feature_importance = feature_importance.sort_values('importance', ascending=True)

# 교차 평가를 위한 RMSE 스코어와 함수
rmse_scorer = make_scorer(mean_squared_error, squared=False)

# 중요도가 낮은 Feature부터 하나씩 제거하면서 모델 성능 평가
rmse_list = []
features_list = []

for i in range(len(feature_importance)):
    selected_features = feature_importance['feature'][i:].tolist()
    X_train_selected = X_train[selected_features]

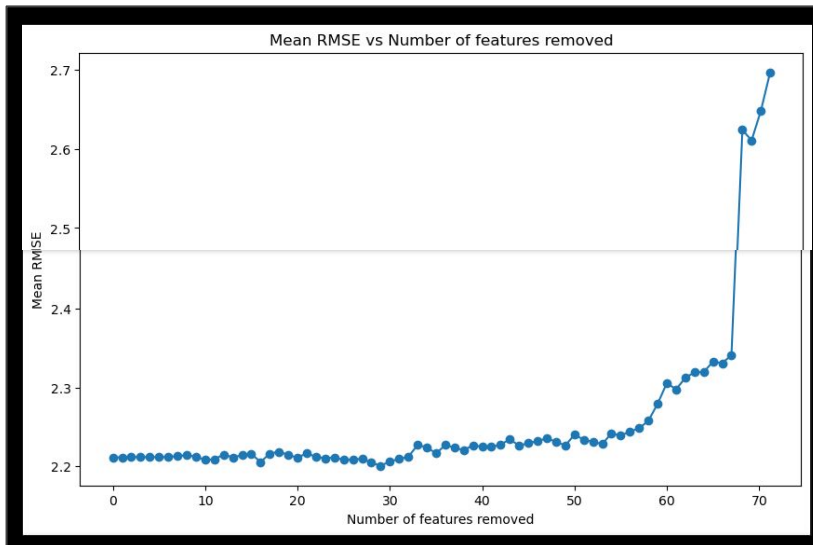
    # 교차 평가를 사용하여 모델 성능 평가
    scores = cross_val_score(model, X_train_selected, y_train, cv=5, scoring=rmse_scorer)
    mean_rmse = np.mean(scores)

    rmse_list.append(mean_rmse)
    features_list.append(selected_features)

    print(f"Removed {i} features, Mean RMSE: {mean_rmse}")

# 최적의 Feature 조합 찾기
min_rmse_index = np.argmin(rmse_list)
best_features = features_list[min_rmse_index]
best_rmse = rmse_list[min_rmse_index]

print(f"Best Mean RMSE: {best_rmse} with features: {best_features}")
```



중요도가 낮은 Feature부터 삭제



2차 모델링

Feature Selection으로 best_var1, best_var2 생성

var1 => best_var1

```
best_var1 = ['bus_route_id2', '6~7_takeoff', 'adult', '1820_w_mean', 'dis_po', '69b', 'teen', '9~10_all_ride_number',  
            'station_code2', '912b', '1820_r_mean', 'kids', '7~8_all_ride_number', 'latitude', 'dis_jejuairport',  
            '9~10_takeoff', 'longitude', '10~9_all_ride_number', '10~11_takeoff', '6~7_ride', '기온(°C)', '11~12_takeoff',  
            '1012b', 'takeoff_sum', '68a', '7~8_ride', '810b', 'dis seongsan', '간수필(mm)', '1820_s_mean', '8~9_takeoff',  
            '8~9_ride', '69a', '9~10_ride', '810a', 'ride_sum', '10~11_ride', '10~11_all_ride_number', '6~7_all_ride_number',  
            '1012a', '912a', '11~12_ride', '1820_rs_mean', 'elder']
```

var2 => best_var2

```
input_var3 = ['sum_avg_spend', '68a', '810a', 'si_제주시', 'dong2', 'bus_interval', 'dis_jejuairport', 'ride_sum',  
            'takeoff_sum', '1820_rs_mean', '1820_rs_sum', '1820_r_mean', '1820_r_sum', '1820_s_mean',  
            '1820_s_sum', 'congestion', 'bus_route_id2', '6~7_all_ride_number', '7~8_all_ride_number',  
            '8~9_all_ride_number', '1012a_mean', '1012b_sum', '10~11_ride_sum', '10~11_takeoff_sum', '11~12_ride_sum',  
            '11~12_takeoff_sum', '1820_r_mean_sum', '1820_r_mean_mean',  
            '1820_r_sum_sum', '1820_r_sum_mean', '1820_rs_mean_sum',  
            '1820_s_mean_sum', '1820_s_mean_mean', '1820_s_sum_sum',  
            '1820_s_sum_mean', '1820_w_mean_sum', '1820_w_mean_mean',  
            '1820_w_sum_mean', '68a_sum', '68a_mean', '68b_sum', 'in_out', 'latitude', 'longitude',  
            '6~7_ride', '7~8_ride', '8~9_ride', '9~10_ride', '10~11_ride', '11~12_ride', '6~7_takeoff', '7~8_takeoff',  
            '8~9_takeoff', '9~10_takeoff', '10~11_takeoff', '11~12_takeoff',  
            'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4',  
            'weekday_5', 'weekday_6', 'dis_jejusi', '68b_mean', '6~7_ride_sum',  
            '6~7_ride_mean', '6~7_takeoff_sum', '6~7_takeoff_mean', '7~8_ride_sum',  
            '7~8_ride_mean', '7~8_takeoff_sum', '7~8_takeoff_mean', '810a_sum',  
            '810b_sum', '8~9_ride_sum', '8~9_takeoff_sum', '8~9_takeoff_mean',  
            '9~10_ride_sum', '9~10_takeoff_sum', 'route_station_weekday2']
```

*var2는 Feature Selection 후 오히려 RMSE가 증가하여
var2 = best_var2로 설정함



2차 모델링

4개의 모델 생성 후 best_var1, best_var2 학습

best_var1 ➡ LightGBM(GridSearchCV)

best_var1 ➡ RandomForest(GridSearchCV)

best_var2 ➡ LightGBM(GridSearchCV)

best_var2 ➡ RandomForest(GridSearchCV)



2차 모델링

4개의 모델 생성 후 best_var1, best_var2 학습

best_var1 ➡ LightGBM(GridSearchCV) ➡ 2.2178664654

best_var1 ➡ RandomForest(GridSearchCV) ➡ 2.2974108683

best_var2 ➡ LightGBM(GridSearchCV) ➡ 2.227398048

best_var2 ➡ RandomForest(GridSearchCV) ➡ 2.2008937695



2차 모델링

4개의 모델 생성 후 best_var1, best_var2 학습

best_var1 ➡ LightGBM(GridSearchCV) ➡ 2.2178664654

best_var1 ➡ RandomForest(GridSearchCV) ➡ 2.2974108683

best_var2 ➡ LightGBM(GridSearchCV) ➡ 2.227398048

best_var2 ➡ RandomForest(GridSearchCV) ➡ **2.2008937695**

↑
2차 모델링 최저 RMSE



3차 모델링

4개 모델의 Optuna 하이퍼파라미터최적화

```
import optuna
from sklearn import lgb
from sklearn.model_selection import cross_val_score

def objective(trial):
    param = {
        "boosting_type": "gbdt",
        "num_leaves": 100,
        "min_child_samples": 10,
        "min_child_weight": 0.1,
        "max_depth": 5,
        "max_bin": 255,
        "feature_fraction": 0.8,
        "lambda": 1e-4,
        "n_estimators": trial.suggest_int("n_estimators", 10, 100),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 5, 10),
        "min_data_in_node": trial.suggest_int("min_data_in_node", 5, 10),
        "min_split_gain": trial.suggest_float("min_split_gain", 0.0, 0.1),
        "monotone_constraints": trial.suggest_categorical("monotone_constraints", ["", "C", "D"]),
        "num_threads": 4,
        "verbose": -1,
    }
    scores = cross_val_score(model, X_train(X_train), y_train, cv=5, scoring="neg_mean_squared_error")
    return scores.mean()

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)

print("Best hyperparameters:", study.best_params)
print("Best RMSE:", study.best_value)

best_model = lgb.LGBMRegressor(**study.best_params)
best_model.fit(X_train(X_train), y_train)
X_test(X_test)
y_test(y_test)
print("RMSE on test set:", study.best_value)
```

LightGBM_best_var1

+

Optuna

```
import optuna
from sklearn import lgb
from sklearn.model_selection import cross_val_score

def objective(trial):
    param = {
        "boosting_type": "gbdt",
        "num_leaves": 100,
        "min_child_samples": 10,
        "min_child_weight": 0.1,
        "max_depth": 5,
        "max_bin": 255,
        "feature_fraction": 0.8,
        "lambda": 1e-4,
        "n_estimators": trial.suggest_int("n_estimators", 10, 100),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 5, 10),
        "min_data_in_node": trial.suggest_int("min_data_in_node", 5, 10),
        "min_split_gain": trial.suggest_float("min_split_gain", 0.0, 0.1),
        "monotone_constraints": trial.suggest_categorical("monotone_constraints", ["", "C", "D"]),
        "num_threads": 4,
        "verbose": -1,
    }
    scores = cross_val_score(model, X_train(X_train), y_train, cv=5, scoring="neg_mean_squared_error")
    return scores.mean()

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)

print("Best hyperparameters:", study.best_params)
print("Best RMSE:", study.best_value)

best_model = lgb.LGBMRegressor(**study.best_params)
best_model.fit(X_train(X_train), y_train)
X_test(X_test)
y_test(y_test)
print("RMSE on test set:", study.best_value)
```

LightGBM_best_var2

+

Optuna

```
import optuna
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error

def objective(trial):
    param = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 100),
        "max_depth": trial.suggest_int("max_depth", 5, 10),
        "min_child_samples": trial.suggest_int("min_child_samples", 5, 10),
        "min_child_weight": trial.suggest_float("min_child_weight", 0.0, 0.1),
        "min_split_gain": trial.suggest_float("min_split_gain", 0.0, 0.1),
        "max_features": trial.suggest_categorical("max_features", ["auto", "best", "sqrt"]),
    }
    scores = cross_val_score(model, X_train(X_train), y_train, cv=5, scoring="neg_mean_squared_error")
    return scores.mean()

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)

print("Best hyperparameters:", study.best_params)
print("Best RMSE:", study.best_value)
```

RandomForest_best_var1

+

Optuna

```
import optuna
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error

def objective(trial):
    param = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 100),
        "max_depth": trial.suggest_int("max_depth", 5, 10),
        "min_child_samples": trial.suggest_int("min_child_samples", 5, 10),
        "min_child_weight": trial.suggest_float("min_child_weight", 0.0, 0.1),
        "min_split_gain": trial.suggest_float("min_split_gain", 0.0, 0.1),
        "max_features": trial.suggest_categorical("max_features", ["auto", "best", "sqrt"]),
    }
    scores = cross_val_score(model, X_train(X_train), y_train, cv=5, scoring="neg_mean_squared_error")
    return scores.mean()

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)

print("Best hyperparameters:", study.best_params)
print("Best RMSE:", study.best_value)
```

RandomForest_best_var2

+

Optuna

3차 모델링

8개의 모델 생성 후 A, B, C, D 생성

best_var1 → LightGBM(GridSearchCV)
 → LightGBM(Optuna)

best_var1 → LightGBM(GridSearchCV)
 → LightGBM(Optuna)

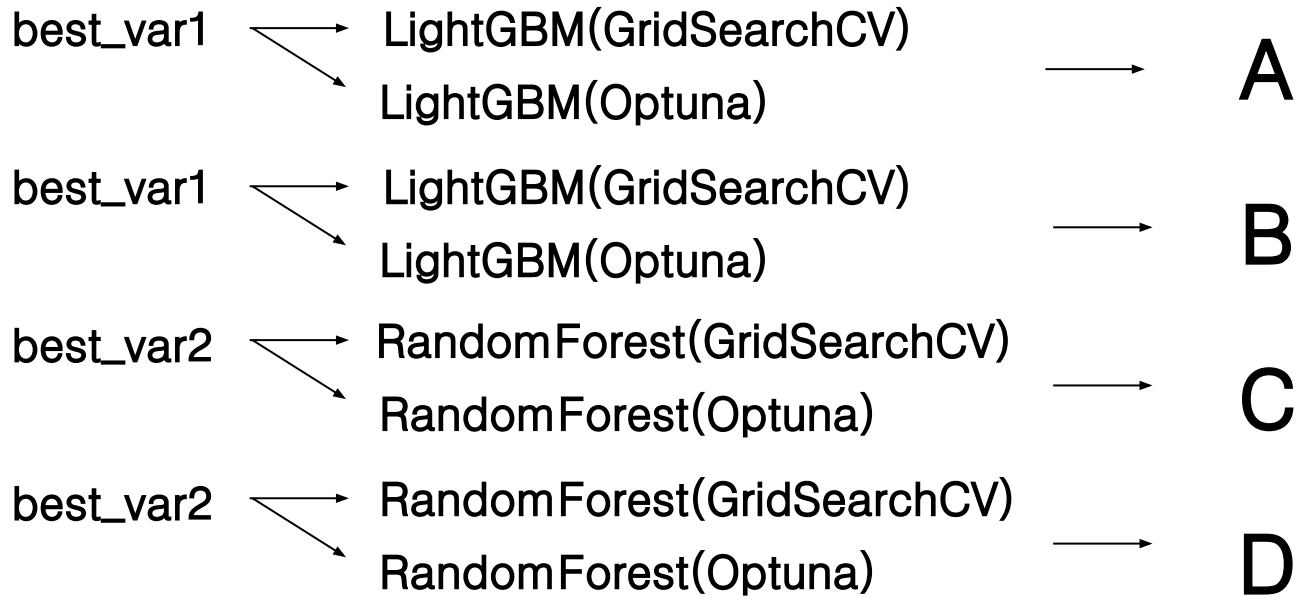
best_var2 → RandomForest(GridSearchCV)
 → RandomForest(Optuna)

best_var2 → RandomForest(GridSearchCV)
 → RandomForest(Optuna)



3차 모델링

8개의 모델 생성 후 A, B, C, D 생성



3차 모델링

RMSE가 가장 낮은 조합 찾기

1개 조합 : (A, B, C, D) >> 4개

2개 조합 : (A+B, A+C, A+D, B+C, B+D, C+D) / 2 >> 6개

3개 조합 : (A+B+C, A+B+D, B+C+D, A+C+D) / 3 >> 4개

4개 조합 : (A+B+C+D) / 4 >> 1개

$$A = 2.2178664654$$

$$A + B = 2.2417782747$$

$$A + B + C = 2.2256394242$$

$$A + B + C + D = 2.2088098707$$

$$B = 2.2178664654$$

$$A + C = 2.2122613982$$

$$A + B + D = 2.2124034029$$

$$C = 2.227398048$$

$$A + D = 2.193481764$$

$$B + C + D = 2.2125531226$$

$$D = 2.2008937695$$

$$B + C = 2.2394690628$$

$$A + C + D = 2.1967870071$$

$$B + D = 2.2218119164$$

$$C + D = 2.1985890093$$



3차 모델링

RMSE가 가장 낮은 조합 찾기

1개 조합 : (A, B, C, D) >> 4개

2개 조합 : (A+B, A+C, A+D, B+C, B+D, C+D) / 2 >> 6개

3개 조합 : (A+B+C, A+B+D, B+C+D, A+C+D) / 3 >> 4개

4개 조합 : (A+B+C+D) / 4 >> 1개

$$A = 2.2178664654$$

$$A + B = 2.2417782747$$

$$A + B + C = 2.2256394242$$

$$A + B + C + D = 2.2088098707$$

$$B = 2.2178664654$$

$$A + C = 2.2122613982$$

$$A + B + D = 2.2124034029$$

$$C = 2.227398048$$

$$A + D = 2.193481764$$

$$B + C + D = 2.2125531226$$

$$D = 2.2008937695$$

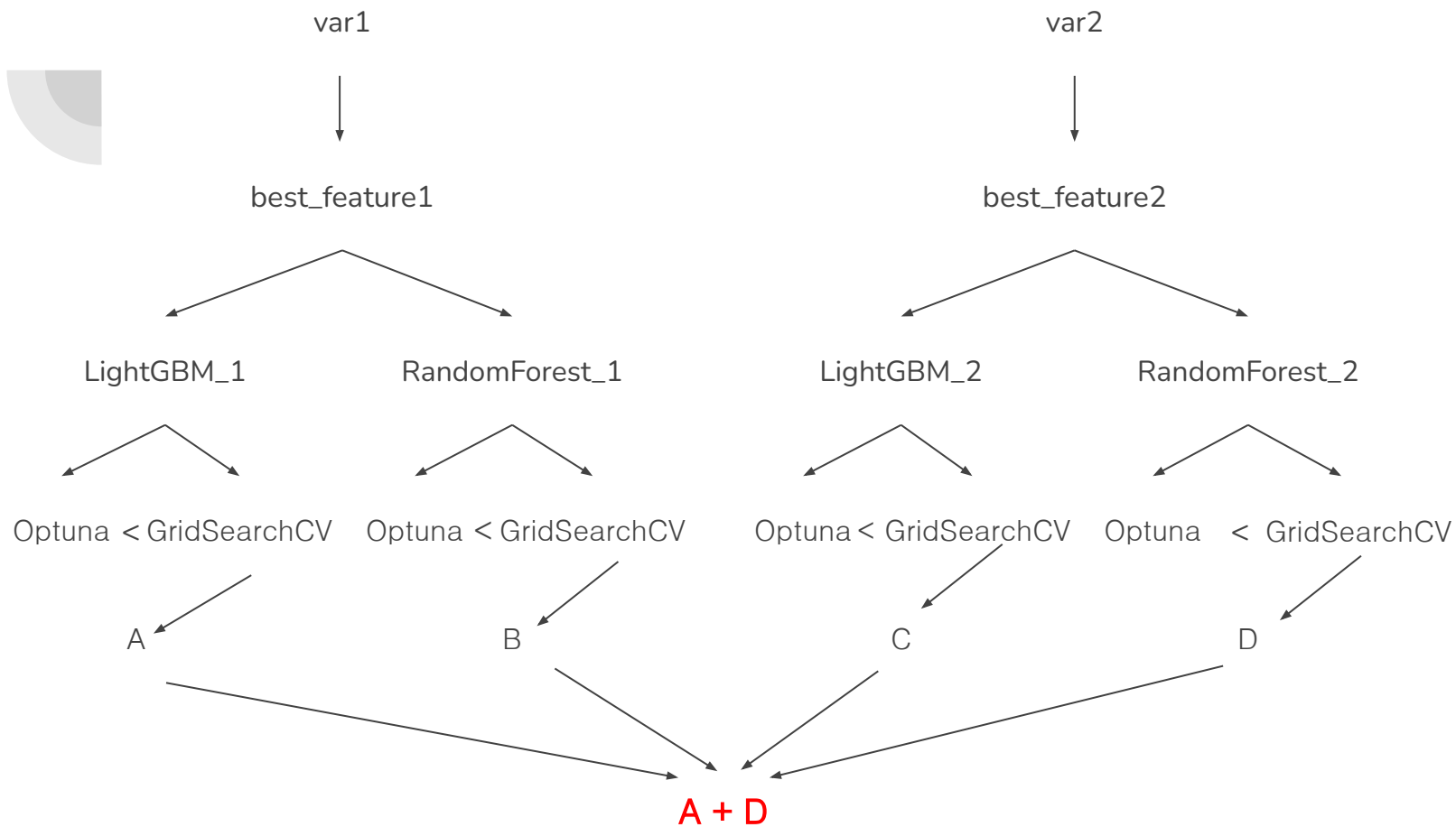
$$B + C = 2.2394690628$$

$$A + C + D = 2.1967870071$$









$$B + D = 2.2218119164$$

$$C + D = 2.1985890093$$





결과

1	i7_돌하르방	   	2.15681	38
2	Dining	  	2.1609	68
3	TEAM-EDA	    	2.16185	27
4	STATION		2.16739	3
5	임진혁(JinSloth)		2.17037	6
6	KIMCHAJANG	  	2.17543	33
7	sayho		2.17899	2
8	GGg123	 	2.18337	15
9	내가일등할거다		2.1917	70
10	unun		2.1933	11
11	우리팀 ^^		2.19348	
11	LEE91		2.19658	6



Q & A

