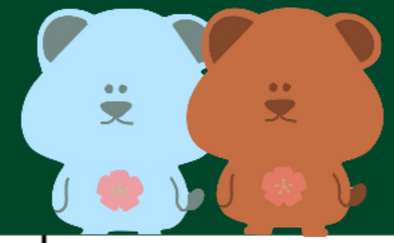


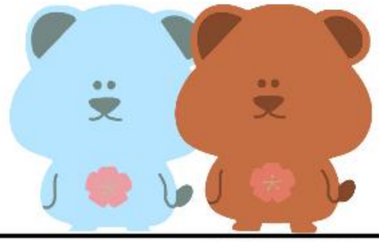
서울특별시 퇴근시간 승차인원 예측

Euron 입문 초급 세션

버스타요팀 - 이채원 정지은 조승연

EWHA EURON

목차



EWHA EURON

#01 프로젝트 주제

1) 프로젝트 주제

서울특별시 퇴근 시간(18~20) 버스 승차인원 예측

2) 주제 선정 이유

a. 정류장별 승차 인원을 미리 예측하면 <명동 버스 대란>과 같은 버스 대란을 피할 수 있음

b. 데이콘의 <제주도 퇴근시간 버스 승차인원 예측> 대회와 달리 서울특별시 데이터를 사용하여 전처리 전체 과정을 직접 진행하고, 학습시키는 과정에서 응용력을 높일 수 있음

3) 기대 효과

버스 정류장의 특정 노선에 탑승하는 승차 인원을 예측하여 아래와 같은 효과를 기대할 수 있음

a. 정류장 위치를 분산 → 버스의 효율적인 운행 가능

b. 출퇴근 시간 버스 이용객의 편의성 향상

프로젝트 진행 과정

EWHA EURON

#2 EDA

EWHA EURON

#02 EDA

사용 데이터

a. 서울특별시 버스 정류장별 노선별 승하차 인원 데이터
b. 서울특별시 버스 정류장 위도, 경도 데이터
c. 서울특별시 월별 날씨 데이터

데이터 구조

서울 특별시 정류장별 노선별 승하차 인원 데이터

	id	month	bus_id	bus_route_id	station_code	station_name	6~7_ride	6~7_takeoff	7~8_ride	7~8_takeoff	...	17~18_takeoff	18~19_rid
0	0	202301	741	100000001	1001	종로2가사거리 (00077)	140	62	509	300	...	462	31
1	1	202301	470	100000001	1001	종로2가사거리 (00067)	194	59	558	271	...	545	33

서울특별시 정류장 위도, 경도 데이터

	정류장_ID	정류장_명칭	정류장_유형	정류장_번호	위도	경도	버스도착정보안내기_설치_여부
0	100000001	종로2가사거리	중앙차로	1001.0	37.569806	126.987752	설치
1	100000002	창경궁.서울대학교병원	중앙차로	1002.0	37.579433	126.996521	설치

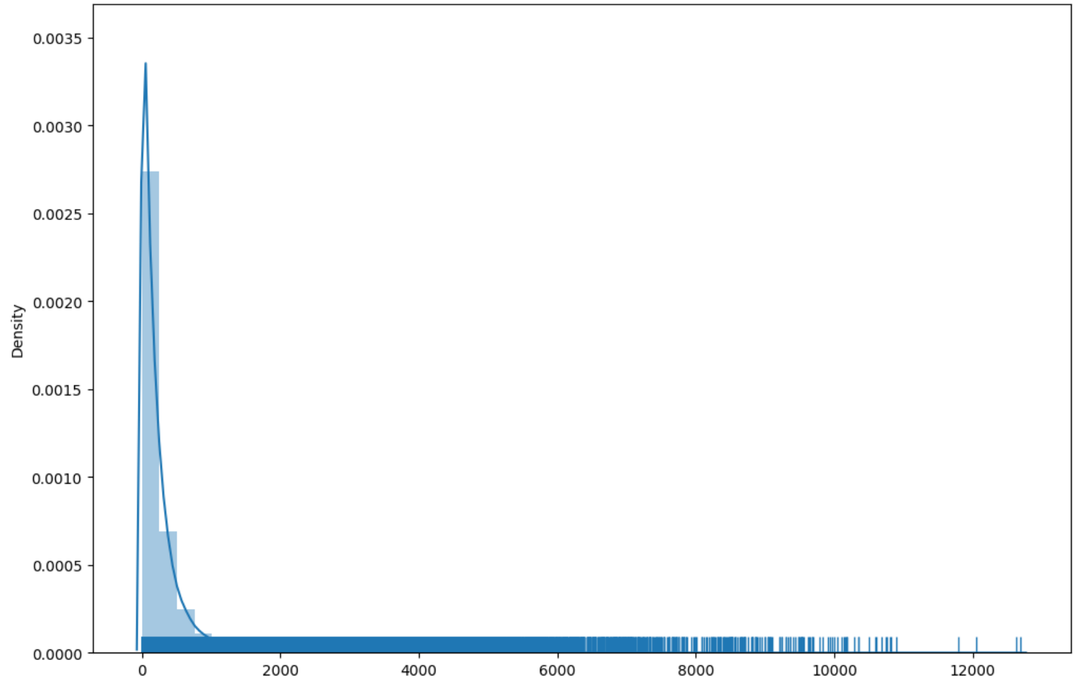
월별 날씨 데이터

	월	강수량(mm)	평균기온(℃)
0	1	47.9	-1.5
1	2	1.0	2.3

#02 EDA

승하차 데이터

주요 퇴근시간 17~20시 승차 분포 확인
→ 0인 데이터가 많은 것을 확인



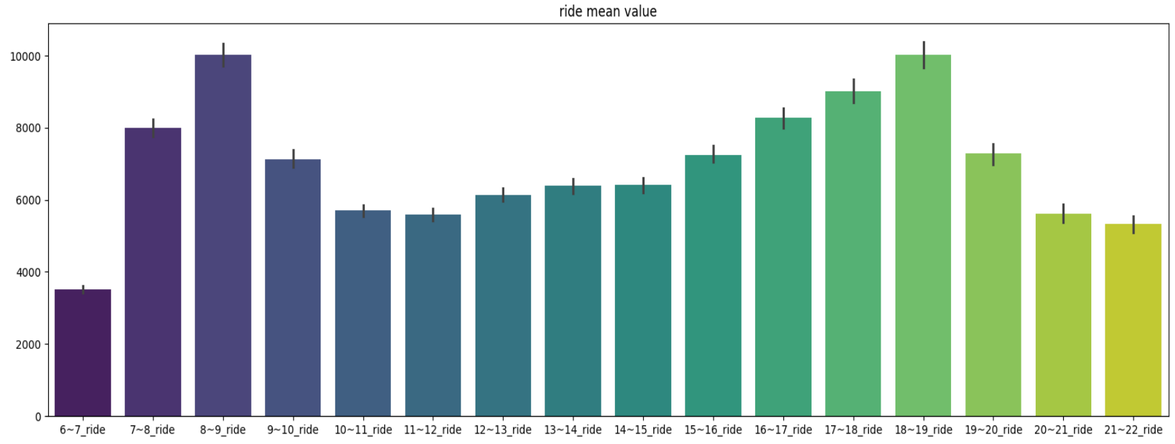
승하차 데이터가 1:1 매핑되지 않음
서울시 외 하차하는 경우는 집계되지 않았기 때문

```
승차 칼럼 총 합
min        0
max    10791
dtype: int64
======================
하차 칼럼 총 합
min        0
max    9942
dtype: int64
======================
승하차 칼럼 총 합
min        0
max    43464
dtype: int64
```

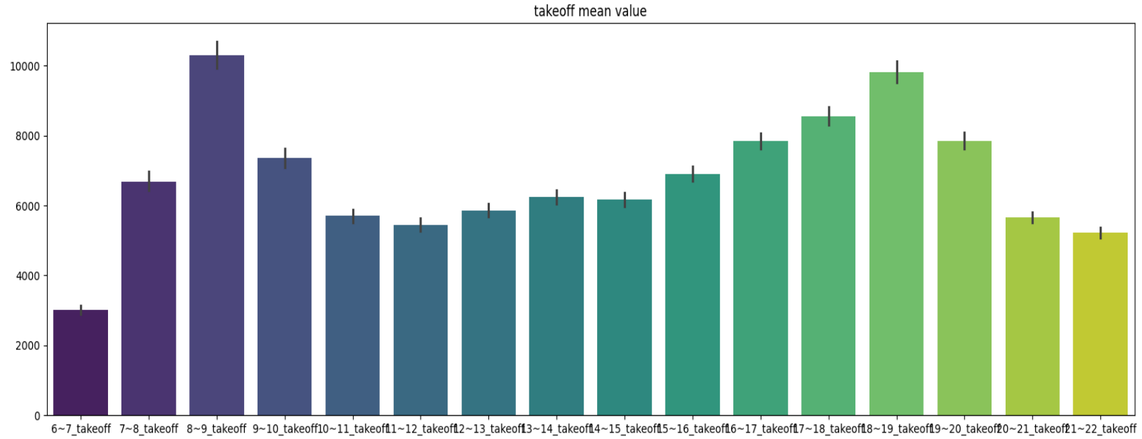
#02 EDA

시간대별 승하차 인원 분석

가장 많은 승차 인원: 8~9시 → 출근 시간



가장 많은 하차 인원: 18-19시 → 퇴근 시간



➡️ 탑승인원 증감 양상이 승,하차에서 거의 비슷하게 나타남

#02 EDA

버스 정류장별 퇴근 시간 승차 인원 분석

승차 인원 합계를 기준으로 가장 붐비는 top20

statation_code를 기준으로 승하차 데이터와 매핑하여, 정류장 위치 데이터를 불러옴

이후 지도 API 를 사용하여 해당 정류장의 위치를 지도에 나타냄

➡️ 강남, 구로 등 직장이 많은 위치에서 승차가 많이 일어나는 것을 확인

```
import folium
from folium.plugins import MarkerCluster

# 서울 주요 장소의 임의 지역 위도, 경도
seoul = (37.5665, 126.9780)  # 서울 중심지

# 서울 지역이 보일 수 있는 위치의 위도, 경도를 표시한 뒤, folium.Map에 변수로 넣고, map_osm에 할당
map_osm = folium.Map(location=seoul, zoom_start=11)
mc = MarkerCluster()

mc.add_child(folium.Marker(location=seoul, popup='서울 중심지', icon=folium.Icon(color='red', icon='info-sign')))
map_osm.add_child(mc)

for row in top_takeoff_station.itertuples():
    mc.add_child(folium.Marker(location=[row.lat,  row.lng], popup=row.station_name)) #마커 생성
    map_osm.add_child(mc) #마커를 map_osm에 추가

map_osm
```

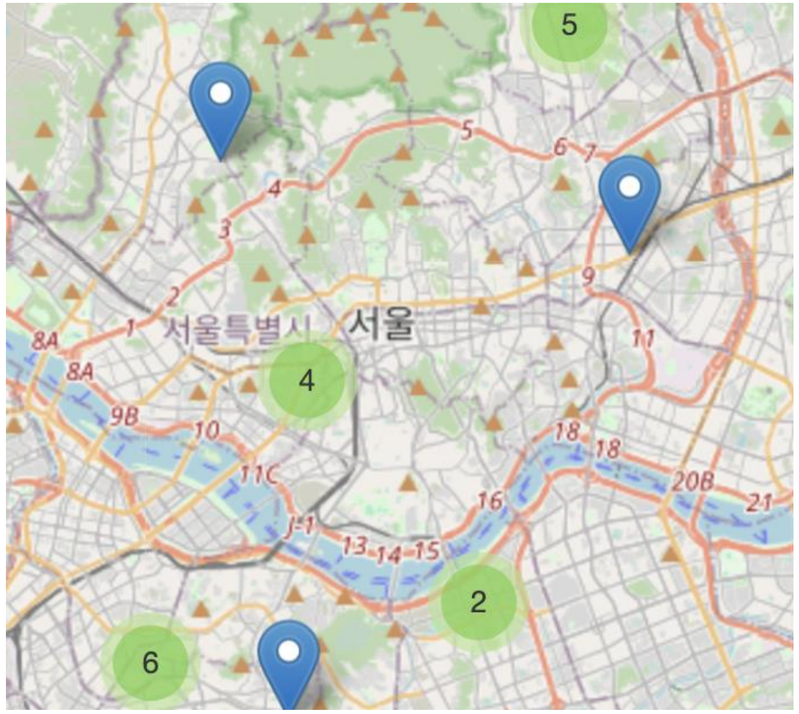
#02 EDA

버스 정류장별 출퇴근 시간 하차 인원 분석

승차 인원 분석과 같은 방식으로 진행

승차 분석과 유사하게 환승센터, 구로, 고속 터미널 주변 정류장에서 많은 하차가 발생함

	station_name	station_code	lat	lng
795	순천향대학병원.한남오거리	3165.0	37.535491	127.005729
1963	청량리역환승센타	6015.0	37.580403	127.045431
3389	수유역.강북구청	9004.0	37.637592	127.025308
3397	미아사거리역	9012.0	37.612894	127.030213
3561	롯데백화점미아점	9277.0	37.614709	127.030756
3591	수유(강북구청)역	9013.0	37.638290	127.025870
3892	쌍문역	10015.0	37.648817	127.034769



#02 EDA

버스 노선별 퇴근 시간 승차 인원 분석

노선 종류에 따라 노선별 승하차 인원 분석

마을, 시내(간선, 지선, 순환), 광역 버스로 나눠서 진행

마을 버스 승차 top10

	bus_id	main_ride
475	동작01	858798
444	금천03	786486
441	금천01	730154
550	서초18	680788
592	양천01	636544
460	노원14	567722
610	용산02	558653
407	강서05	547058
516	서대문03	509462
482	동작08	493616

시내 버스 승차 top10

	bus_id	main_ride
67	143	1733208
76	160	1621321
119	272	1455465
118	271	1439343
73	150	1397286
63	130	1366886
209	5531	1308810
142	340	1244812
78	163	1244368
75	152	1235722

광역 버스 승차 top10

	bus_id	main_ride
368	9401	453134
376	9707	203469
374	9701	195829
377	9711	186013
371	9404	143146
370	9403	83409
372	9408	79418
369	9401-1	62220
378	9714	42339
375	9703	40938

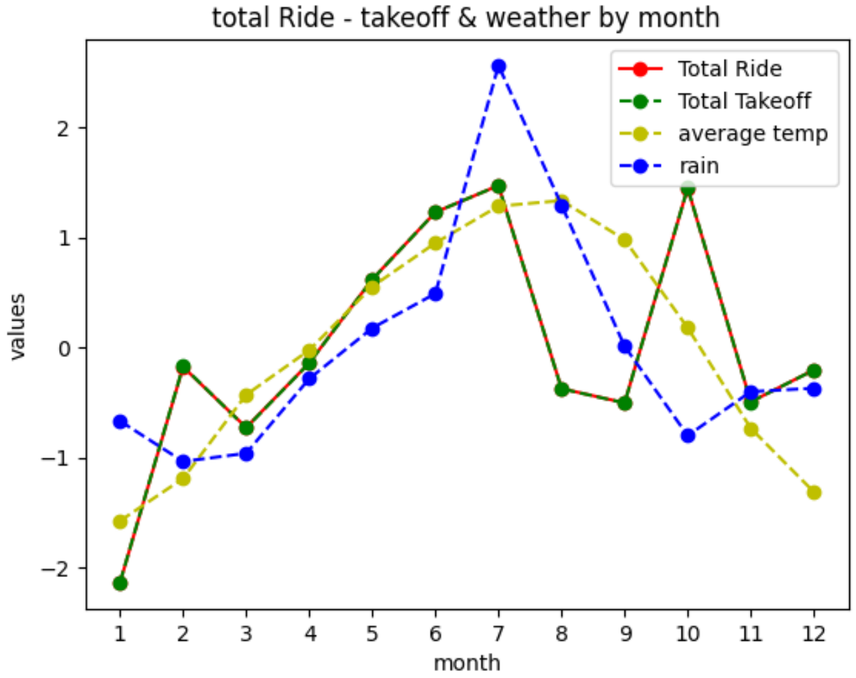
#02 EDA

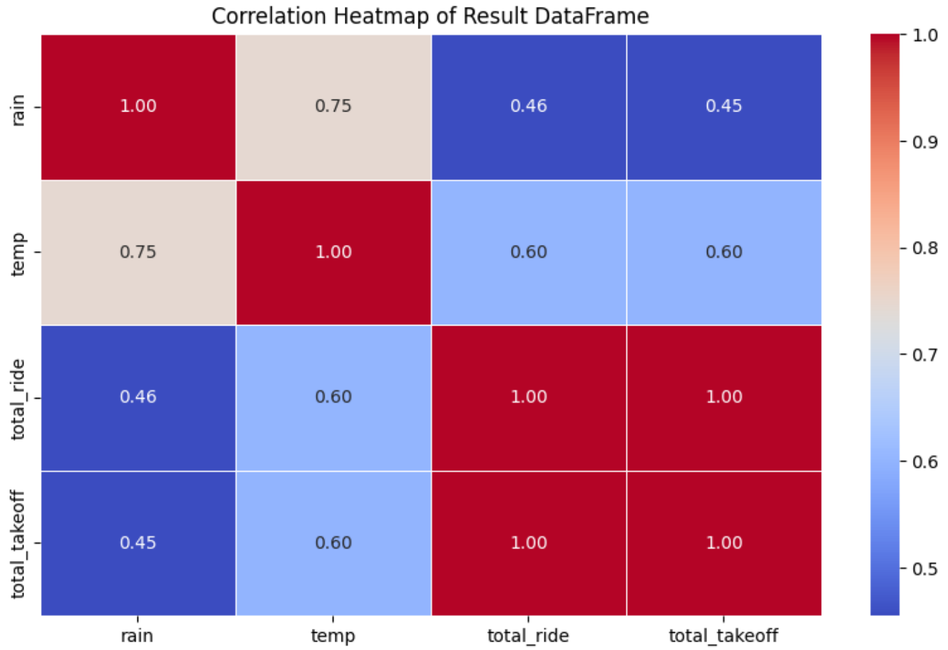
날씨와 버스 승하차 인원 상관 관계 분석

서울시 월별 기온, 강수량 데이터와 월별로 합산된 버스 승하차 데이터 사용

달별 승/하차 총합과 평균 기온, 강수량을 정규화하고, 꺾은선 그래프와 Heat map으로 상관 관계를 분석

➡️ 날씨와 승하차 인원의 상관 관계가 크지 않은 것으로 판단





#3 전처리

EWHA EURON

#03 전처리

사용 데이터

Train – 2022년 9월 ~2023년 8월 승하차 데이터
Test – 2023년 9월 ~ 2023년 11월 승하차 데이터

1. 데이터 로딩

2. 불필요한 피처 삭제

등록일자, 교통수단타입코드, 버스정류장ARS번호, 노선명 칼럼 삭제

3. 퇴근 시간 이후 시간대 삭제

퇴근시간 이전 승하차 데이터로 예측하므로, 20시 이후 승하차 인원 칼럼을 삭제

4. 칼럼명 영문 변환

학습에 용이하도록 칼럼명을 전부 영어로 변환

#03 전처리

6. object 타입 피처 처리

Object 타입으로 구성된 피처를 모델이 학습할 수 있는 데이터로 대체

a. bus_station_name (역명) 처리

역명 뒤 5자리+ 표준 정류장 ID를 PK로 사용함

예) 1bus_station_ID(역명 뒤 5자리) = 00077, bus_station_SID(표준 정류장ID) = 100000001
-> bus_station_PK = 00077100000001

b. bus_ID (버스노선명) 처리

110A, 종로09와 같이 숫자와 문자열이 함께 섞여 있음

노선명 대신 노선을 구분하기 위해서, 외부 데이터에서 버스노선 고윳값인 route_ID를 가져와 대체함

c. type (버스 타입) 처리

서울간선버스(0), 서울지선버스(1), 서울마을버스(2), 서울순환버스(3), 서울광역버스(4) 숫자로 변경

#03 전처리

7. route_ID 결측치 처리

Object 타입 피처 중 bus_ID (버스 노선명)을 route_ID (버스 노선 고유 번호)로 대체 하였으나,

2022 데이터를 사용하기 때문에 2023 기준으로 노선이 폐지된 경우, 고유 번호 조회가 불가한 일부 노선에서 결측치가 발생함

➡️ 이전 데이터를 검색하여 직접 찾아넣고, 조회 불가한 값의 경우 111110000~111110011 사이 숫자 부여

```
route_ID 값이 null인 bus_ID:
        bus_ID
37            1
174        9703
204         153
540        9403
1989       성동07
2006         17
2236        411
3587       성동06
5412       2235
6048       2234
7274       8112
7655       1156
7661       노원04
```

```
null_routes_df = train[train['route_ID'].isnull()][['bus_ID']]
print("route_ID 값이 null인 bus_ID:")
print(null_routes_df)
train.drop(['bus_ID'], axis=1, inplace=True)
```

```
route_ID 값이 null인 bus_ID:
Empty DataFrame
Columns: [bus_ID]
Index: []
```

#4 사전 모델링

EWHA EURON

#04 사전 모델링

카테고리형 변수를 별도의 처리 없이 모델링 진행

1. 스케일링

minMax, standard, robust 3가지 스케일러 사용, target을 18~20 (퇴근 시간) ride로 분리

```python
# 스케일링 함수 정의
def scaler(type, train, test):
    if type == 'standard':
        scaler = StandardScaler()
    elif type == 'minmax':
        scaler = MinMaxScaler()
    elif type == 'robust':
        scaler = RobustScaler()

    # 입력 받은 타입에 맞는 스케일러로 스케일링
    train_scaled = pd.DataFrame(data=scaler.fit_transform(train), columns=train.columns)
    test_scaled = pd.DataFrame(data=scaler.fit_transform(test), columns=test.columns)
    train_scaled = train_scaled.astype('float')
    test_scaled = test_scaled.astype('float')

    # 타겟값 분리
    X_train = train_scaled.drop(columns=['18~20_ride'])
    y_train = train_scaled['18~20_ride']
    X_test = test_scaled.drop(columns=['18~20_ride'])
    y_test = test_scaled['18~20_ride']

    return X_train, X_test, y_train, y_test
```

#04 사전 모델링

2. Lgbm regressor 모델 학습 및 평가

lgbm_regressor 모델을 사용하여, 학습하고 MSE, R2 score를 통해 평가 진행

각 스케일러로 스케일링한 데이터를 학습한 결과를 확인함

```python
def lgbm_modeling(X_train, X_test, y_train, y_test, type):
    # 학습 데이터를 학습 및 검증 데이터로 나누기
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
    lgbm = lgb.LGBMRegressor(num_iterations = 1000, learning_rate = 0.1, metric='l2', force_col_wise=True,
                             n_jobs=-1, verbosity=0)
    # 검증 데이터를 eval_set에 추가
    eval_set = [(X_valid, y_valid)]
    # tqdm을 사용하여 학습 진행률 표시
    for i in tqdm(range(0, len(X_train), 10000)):
        X_batch = X_train[i:i+10000]
        y_batch = y_train[i:i+10000]
        lgbm.fit(X_batch, y_batch, eval_set=eval_set, eval_metric='mse', early_stopping_rounds=10, verbose=False)

        # 각 회차의 MSE 계산
        if i % 10000 == 0:  # 매 10000회차마다 MSE 출력
            y_pred = lgbm.predict(X_test)
            mse = mean_squared_error(y_test, y_pred)
            print(f"Iteration {i}: Test MSE = {mse:.4f}")

    # 테스트 데이터로 예측 수행
    y_pred = lgbm.predict(X_test)
    # 최종 MSE, R2 score 출력
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'LGBM - {type} scaling MSE: {mse:.4f}, R2 Score: {r2:.4f}')

    return lgbm
```

standard scaling
MSE: 0.1870, R2 Score: 0.8130

minmax scaling
MSE: 0.0001, R2 Score: 0.7869

robust scaling
MSE: 0.2472, R2 Score: 0.8090

#04 사전 모델링

3. lgbm regressor 모델 튜닝

lgbm_regressor 모델을 사용하여, 학습하고 MSE, R2 score를 통해 평가 진행

R2 score 측면에서 기존 0.8 수준에서 0.9 후반으로 향상된 것을 확인

```python
def lgbm_tuning(X_train, X_test, y_train, y_test, type):
    # 학습 데이터를 학습 및 검증 데이터로 나누기
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

    # 그리드 서치 대상 파라미터 설정
    param_grid = {
        'num_iterations': [1000], 'learning_rate': [0.1, 0.5], 'max_depth': [3, 5, 7],
        'metric': ['l2'], 'force_col_wise': [True], 'n_jobs': [-1], 'verbosity': [-1]
    }

    # LightGBM Regressor 생성
    lgbm = lgb.LGBMRegressor()

    # GridSearchCV를 사용하여 최적의 하이퍼파라미터 찾기
    grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2
    grid_search.fit(X_train, y_train)

    # 최적의 모델을 찾은 후 학습 데이터에 대해 다시 학습
    best_model = grid_search.best_estimator_
    best_model.fit(X_train, y_train, eval_set=[(X_valid, y_valid)], eval_metric='mse', early_stopping_rounds=10, verbose=False)

    # 테스트 데이터로 예측 수행
    y_pred = best_model.predict(X_test)

    # 최종 MSE, R2 score 출력
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'LGBM - {type} scaling MSE: {mse:.4f}, R2 Score: {r2:.4f}')
```

standard scaling
MSE: 0.0250, R2 Score: 0.9750

minmax scaling
MSE: 0.0000, R2 Score: 0.9637

robust scaling
MSE: 0.0239, R2 Score: 0.9815

#5 모델링 & 튜닝

EWHA EURON

#05 모델링 & 튜닝

1. 인코딩 & 스케일링

.astype(‘category’)를 이용하여, 카테고리형 변수 route_ID, bus_station_PK, type 등 처리

minMax, standard, robust 3가지 종류의 스케일러로 스케일링 진행

```
def cat_scale(scaler, X, X_test):
  if scaler == 'minmax':
    transformer = MinMaxScaler()
  elif scaler == 'standard':
    transformer = StandardScaler()
  elif scaler == 'robust':
    transformer = RobustScaler()

  X_scaled1 = transformer.fit_transform(X[['month', '6~8 ride', '8~10 ride', '10~12 ride', '12~14 ride', '14~16 ride',
                                           '16~18 ride', '6~8 takeoff', '8~10 takeoff', '10~12 takeoff',
                                           '12~14 takeoff', '14~16 takeoff', '16~18 takeoff']])
  X_scaled2 = X.copy()
  X_scaled2[['month', '6~8 ride', '8~10 ride', '10~12 ride', '12~14 ride', '14~16 ride',
                                           '16~18 ride', '6~8 takeoff', '8~10 takeoff', '10~12 takeoff',
                                           '12~14 takeoff', '14~16 takeoff', '16~18 takeoff']] = X_scaled1


  X_scaled3 = transformer.transform(X_test[['month', '6~8 ride', '8~10 ride', '10~12 ride', '12~14 ride', '14~16 ride',
                                           '16~18 ride', '6~8 takeoff', '8~10 takeoff', '10~12 takeoff',
                                           '12~14 takeoff', '14~16 takeoff', '16~18 takeoff']])
  X_scaled4 = X_test.copy()
  X_scaled4[['month', '6~8 ride', '8~10 ride', '10~12 ride', '12~14 ride', '14~16 ride',
                                           '16~18 ride', '6~8 takeoff', '8~10 takeoff', '10~12 takeoff',
                                           '12~14 takeoff', '14~16 takeoff', '16~18 takeoff']] = X_scaled3

  X_scaled2[['route_ID', 'bus_station_PK','type']] = X_scaled2[['route_ID', 'bus_station_PK','type']].astype('category')
  X_scaled4[['route_ID', 'bus_station_PK','type']] = X_scaled4[['route_ID', 'bus_station_PK','type']].astype('category')
  return X_scaled2, X_scaled4
```

#05 모델링 & 튜닝

2. lgbm regressor 모델 학습 및 평가

lgbm_regressor 모델을 사용하여, 학습하고 MSE, RMSE, R2 score를 통해 평가 진행

각 스케일러로 스케일링한 데이터를 학습한 결과를 확인함

```python
def model_eval(model, X_train, X_test, y_train, y_test, scaler):
  X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

  model.fit(X_train, y_train)

  #검증 데이터 적용
  y_preds_tuned1 = model.predict(X_valid)

  mse_tuned1 = mean_squared_error(y_valid, y_preds_tuned1)
  rmse1_tuned1 = np.sqrt(mse_tuned1)
  r2_tuned1 = r2_score(y_valid, y_preds_tuned1)
  print(f'{scaler} 스케일러를 적용한 경우의 평가')
  print('검증 MSE : {0:.3f}   검증 RMSE : {1:.3F}'.format(mse_tuned1 , rmse1_tuned1))
  print()
  print('검증 R2 score : {0:.3f}'.format(r2_tuned1))

  #테스트 데이터 적용
  y_preds_tuned2 = model.predict(X_test)

  mse_tuned2 = mean_squared_error(y_test, y_preds_tuned2)
  rmse1_tuned2 = np.sqrt(mse_tuned2)
  r2_tuned2 = r2_score(y_test, y_preds_tuned2)

  print('테스트 MSE : {0:.3f}   테스트 RMSE : {1:.3F}'.format(mse_tuned2 , rmse1_tuned2))
  print()
  print('테스트 R2 score : {0:.3f}'.format(r2_tuned2))
```

```
standard 스케일러를 적용한 경우의 평가
검증 MSE : 32307.147   검증 RMSE : 179.742

검증 R2 score : 0.955
테스트 MSE : 28500.714   테스트 RMSE : 168.822

테스트 R2 score : 0.959
```

```
minmax 스케일러를 적용한 경우의 평가
검증 MSE : 32102.030   검증 RMSE : 179.170

검증 R2 score : 0.956
테스트 MSE : 28334.011   테스트 RMSE : 168.327

테스트 R2 score : 0.959
```

```
robust 스케일러를 적용한 경우의 평가
검증 MSE : 32752.750   검증 RMSE : 180.977

검증 R2 score : 0.955
테스트 MSE : 28658.803   테스트 RMSE : 169.289

테스트 R2 score : 0.958
```

#05 모델링 & 튜닝

3. Linear regressor 모델 학습 및 평가

Linear regressor 모델을 사용하여, 학습하고 MSE, RMSE, R2 score, 회귀 계수 를 통해 평가 진행

각 스케일러로 스케일링한 데이터를 학습한 결과를 확인함

```
lr = LinearRegression()

lr.fit(X_scaled1 ,y)

y_preds1 = lr.predict(X_scaled1)

mse1 = mean_squared_error(y, y_preds1)
rmse1 = np.sqrt(mse1)

print('Minmax 스케일러를 적용한 경우의 평가')
print('MSE : {0:.3f}    RMSE : {1:.3F}'.format(mse1 , rmse1))
print()
print('R2 score : {0:.3f}'.format(r2_score(y, y_preds1)))
print('절편 :', lr.intercept_)
print('회귀 계수 :', lr.coef_)
```

```
Standard 스케일러를 적용한 경우의 평가
MSE : 82303.336    RMSE : 286.886

R2 score : 0.886
절편 : 499.6162007968114
회귀 계수 : [ 1.18109457e+01  1.43678234e+01 -1.04264686e+02  8.52763843e+01
  1.72519077e+02  2.14015497e+01 -2.43018769e+02 -6.70175159e+01
  1.78540815e+02 -5.33503354e+01 -1.68012709e+02  8.41390855e+00
  9.28380453e+02  4.17789451e+01 -1.84883888e+00  1.27153391e-01
 -1.13516974e+01  5.90386915e+01  4.25768983e+01 -1.14719039e+02
  2.44551464e+01]
```

```
Minmax 스케일러를 적용한 경우의 평가
MSE : 82303.336    RMSE : 286.886

R2 score : 0.886
절편 : -92.18398625671671
회귀 계수 : [ 2.63976744e+01  4.80170886e+02 -3.03364386e+03  2.66114139e+03
  8.70296509e+03  6.86612305e+02 -1.44972942e+04 -2.78622295e+03
  7.74790695e+03 -1.53463962e+03 -5.18476181e+03  2.68858553e+02
  3.08950103e+04  1.60519299e+03 -1.84883888e+00  1.27153391e-01
 -1.13516974e+01  5.90386915e+01  4.25768983e+01 -1.14719039e+02
  2.44551464e+01]
```

#05 모델링 & 튜닝

4. Hyper opt 이용 모델 학습 및 평가

hyperopt 로 모델을 튜닝하고, 앞에서 사용한 model_eval 함수를 사용하여 튜닝 평가

```
def hypertuning(X_train, y_train):
  X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
  lgb_reg_params = {
    'learning_rate':    hp.uniform('learning_rate',0.1,1),
    'max_depth':        hp.choice('max_depth',        np.arange(2, 100, 1, dtype=int)),
    'colsample_bytree': hp.uniform('colsample_bytree',0.4,1),
    'subsample':        hp.uniform('subsample', 0.6, 1),
    'num_leaves':       hp.choice('num_leaves',       np.arange(1, 200, 1, dtype=int)),
    'reg_alpha':        hp.uniform('reg_alpha',0,1),
    'reg_lambda':       hp.uniform('reg_lambda',0,1),
    'n_estimators':     hp.choice('n_estimators',        np.arange(100, 500, 10, dtype=int)), 'random_state':42
    }
  def f(params):
    lgbm = LGBMRegressor(n_jobs=-1,early_stopping_rounds=None,**params)
    score = cross_val_score(lgbm, X_train, y_train, cv=3,scoring='r2',n_jobs=-1).mean()
    return score
  #r2를 기준으로 설정
  trials = Trials()
  result = fmin(
      fn=f,                          # objective function
      space=lgb_reg_params,   # parameter space
      algo=tpe.suggest,              # surrogate algorithm
      max_evals=50,                 # no. of evaluations
      trials=trials                  # trials object that keeps track of the sample results (optional)
      )
  return result
  print(result)
```

```
minmax 스케일러를 적용한 경우의 평가
검증 MSE : 76066.953   검증 RMSE : 275.802

검증 R2 score : 0.895
테스트 MSE : 70486.795   테스트 RMSE : 265.493

테스트 R2 score : 0.898
```

```
standard 스케일러를 적용한 경우의 평가
검증 MSE : 65084.125   검증 RMSE : 255.116

검증 R2 score : 0.910
테스트 MSE : 62857.909   테스트 RMSE : 250.715

테스트 R2 score : 0.909
```

```
robust 스케일러를 적용한 경우의 평가
검증 MSE : 67190.138   검증 RMSE : 259.211

검증 R2 score : 0.907
테스트 MSE : 63469.158   테스트 RMSE : 251.931

테스트 R2 score : 0.908
```

#05 모델링 & 튜닝

5. 제출 파일 생성

최종 목적인 퇴근 시간 (18~20) 버스 정류장 승차인원 예측 파일을 생성

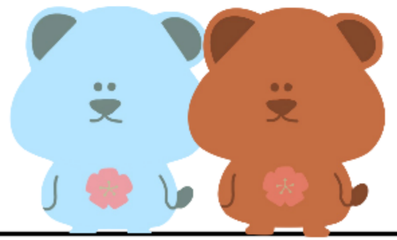
전처리 단계에서 생성한 버스 정류장 고유값인 bus_station_PK를 id로 하여, 조회할 수 있도록 함

```
submission = pd.DataFrame({'id': test['bus_station_PK'],
                           '18~20_ride': y_test, 'minmax':minmax, 'standard':standard, 'robust':robust})
submission
```

	id	18~20_ride	minmax	standard	robust
0	66100000001	720	763.388442	607.065600	664.780032
1	75100000001	867	748.998296	1003.231004	709.439559
2	31100000002	365	591.280746	473.299033	227.233108
3	24100000002	523	655.540669	419.955137	483.188389
4	12100000002	600	849.418156	582.347780	565.338143
...	...	...	...	...	...
113141	81999800002	0	28.676590	24.885587	39.301294
113142	1999800003	4	4.345891	11.596533	1.125851
113143	60999800004	0	4.345891	11.596533	1.125851
113144	127999800005	0	4.345891	29.630701	21.362943
113145	78999800005	0	23.953117	29.630701	21.362943

EWHA EURON

THANK YOU



EWHA EURON