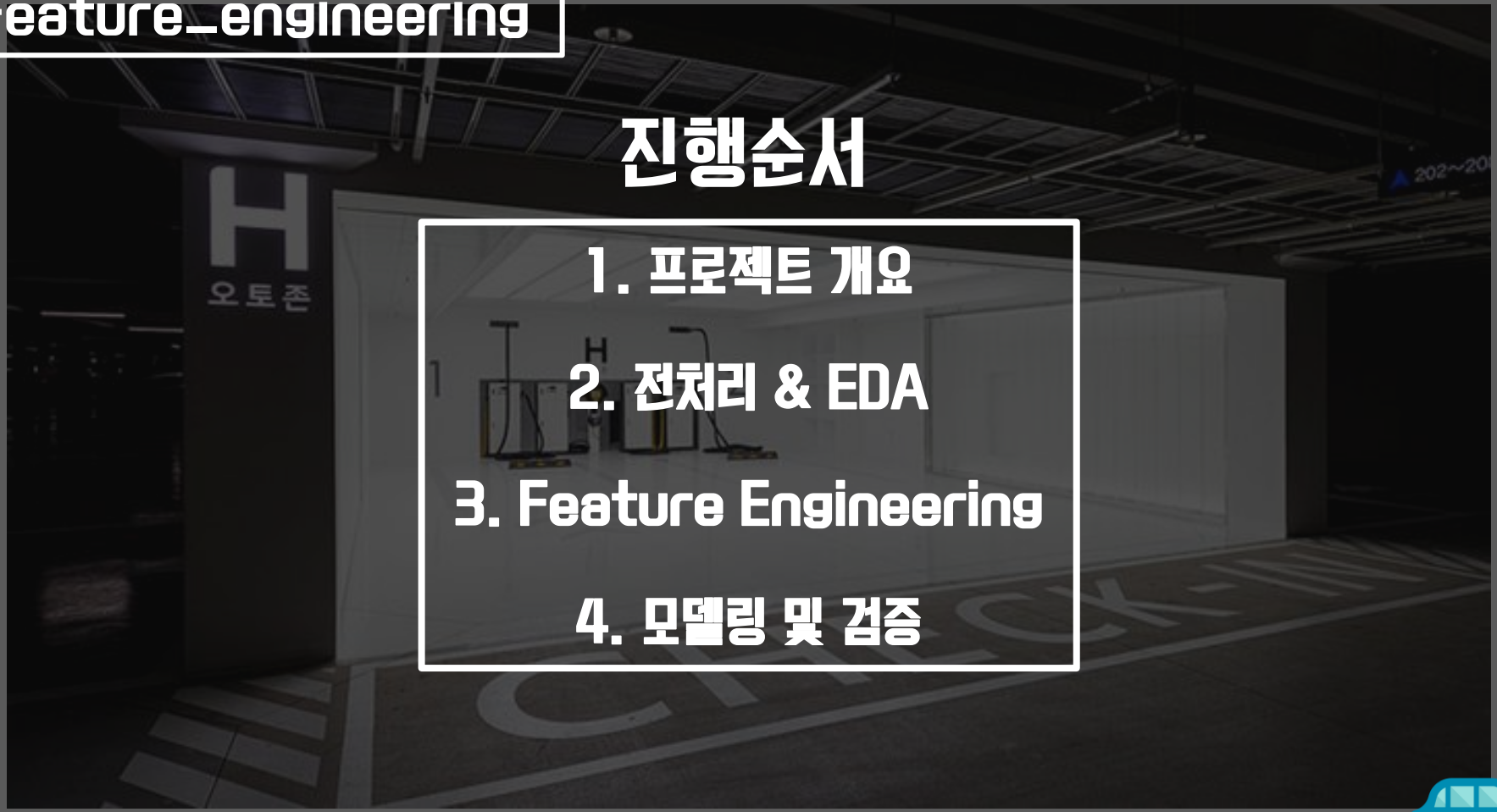

Nb viewer Link : https://nbviewer.jupyter.org/gist/rjsdl56/5c72a87684354f6b00bcdcef3e2b7acc

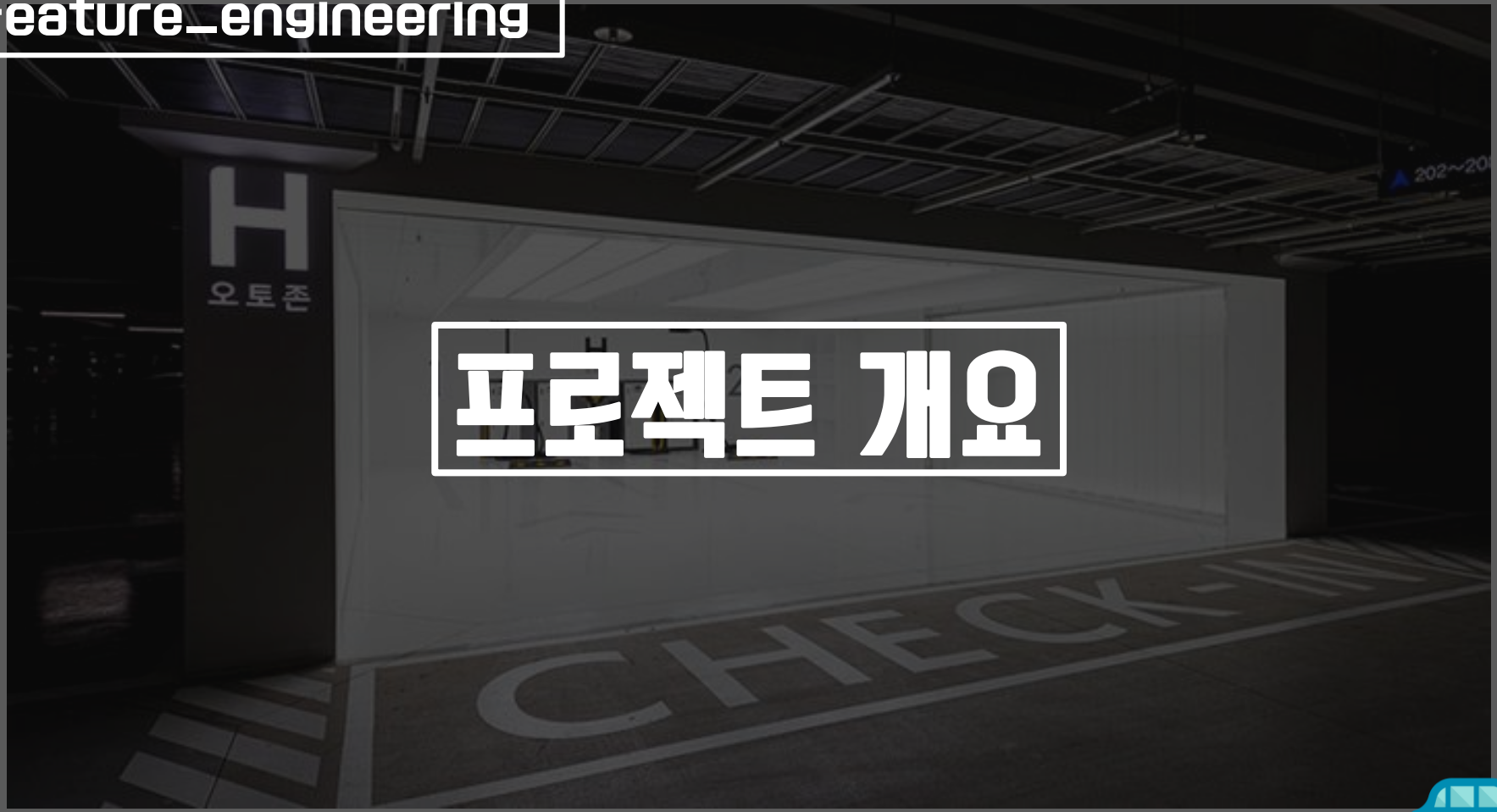
Feature_engineering




# 진행순서

1. 프로젝트 개요
2. 전처리 & EDA
3. Feature Engineering
4. 모델링 및 검증

Feature_engineering




# 프로젝트 개요

# Feature_engineering

## 프로젝트 개요

프로젝트 주제 : 모두컴퍼니 주차장 어플 이용자 마지막 3달 사용건수 예측

프로젝트 기간 : 2021.05.09 ~ 2021.05.15

분석 데이터 :
- 실전db.csv
- 서울시_기상데이터

| | USER_ID | JOIN_DATE | D_TYPE | STORE_ID | GOODS_TYPE | DATE | COUNT | AD1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 2858 | 2014-01-07 | AA | 1892 | A | 2020-01-01 | 1 | GN |
| 1 | 5647 | 2014-02-14 | BB | 182009 | A | 2020-01-01 | 1 | J |
| 2 | 33314 | 2014-11-20 | BB | 82431 | A | 2020-01-01 | 1 | SC |
| 3 | 37001 | 2014-12-04 | BB | 725 | C | 2020-01-01 | 1 | MP |
| 4 | 37819 | 2014-12-07 | AA | 220691 | C | 2020-01-01 | 1 | JRR |

| | 지점 | 일시 | 기온 | 강수량 | 풍속 | 습도 | 일조시간 | 적설량 | 지면온도 | 지중온도 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 108 | 2020-01-01 1:00 | -5.9 | NaN | 1.7 | 40 | NaN | NaN | -2.4 | 3.2 |
| 1 | 108 | 2020-01-01 2:00 | -5.7 | NaN | 0.1 | 42 | NaN | NaN | -2.4 | 3.1 |
| 2 | 108 | 2020-01-01 3:00 | -5.6 | 0.0 | 0.0 | 46 | NaN | NaN | -2.7 | 3.1 |
| 3 | 108 | 2020-01-01 4:00 | -5.4 | NaN | 0.0 | 50 | NaN | NaN | -2.5 | 3.0 |
| 4 | 108 | 2020-01-01 5:00 | -5.2 | NaN | 0.0 | 55 | NaN | NaN | -2.2 | 3.0 |

Feature_engineering


# 전처리 & EDA

# 전처리 & EDA 요약

1. JOIN_DATE를 년도로 변환
2. 유저별 등급별 구분 필요성 확인
3. 주중, 주말, 공휴일을 구분
4. AD1와 STORE_ID 상관관계 확인
5. AD1 내 서울, 비서울 구별 필요성 확인
6. D_TYPE과 GOODS_TYPE 속성 확인

# Feature_engineering

## 전처리 & EDA

### 데이터 확인

```
#데이터 정보 확인
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 879271 entries, 0 to 879270
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   USER_ID      879271 non-null  int64
 1   JOIN_DATE    879271 non-null  object
 2   D_TYPE       879271 non-null  object
 3   STORE_ID     879271 non-null  int64
 4   GOODS_TYPE   879271 non-null  object
 5   DATE         879271 non-null  object
 6   COUNT        879271 non-null  int64
 7   AD1          879271 non-null  object
dtypes: int64(3), object(5)
memory usage: 53.7+ MB
None
```

```
#데이터 결측치 확인
df.isnull()
df.isnull().sum()
```

```
USER_ID       0
JOIN_DATE     0
D_TYPE        0
STORE_ID      0
GOODS_TYPE    0
DATE          0
COUNT         0
AD1           0
dtype: int64
```

```
유니크값 확인 :
column : USER_ID
The number of unique : 165425

column : JOIN_DATE
The number of unique : 2352

column : D_TYPE
The number of unique : 3

column : STORE_ID
The number of unique : 1061

column : GOODS_TYPE
The number of unique : 4

column : DATE
The number of unique : 366

column : COUNT
The number of unique : 56

column : AD1
The number of unique : 85
```

# Feature_engineering



## 데이터 기초 전처리 - 시간

```
# JOIN_DATE와 DATE는 datetime 형태로 바꾼다
df.JOIN_DATE = pd.to_datetime(df.JOIN_DATE)
df.DATE = pd.to_datetime(df.DATE)
```

```
# JOIN_DATE는 년도로 바꾼다.
df.JOIN_DATE = df.JOIN_DATE.dt.to_period(freq="A")
df.JOIN_DATE = df.JOIN_DATE.astype(str)
```

```
# DATE는 월별로 묶어서 새로운 컬럼을 생성
df["MONTH"] = df.DATE.dt.strftime('%m')
```

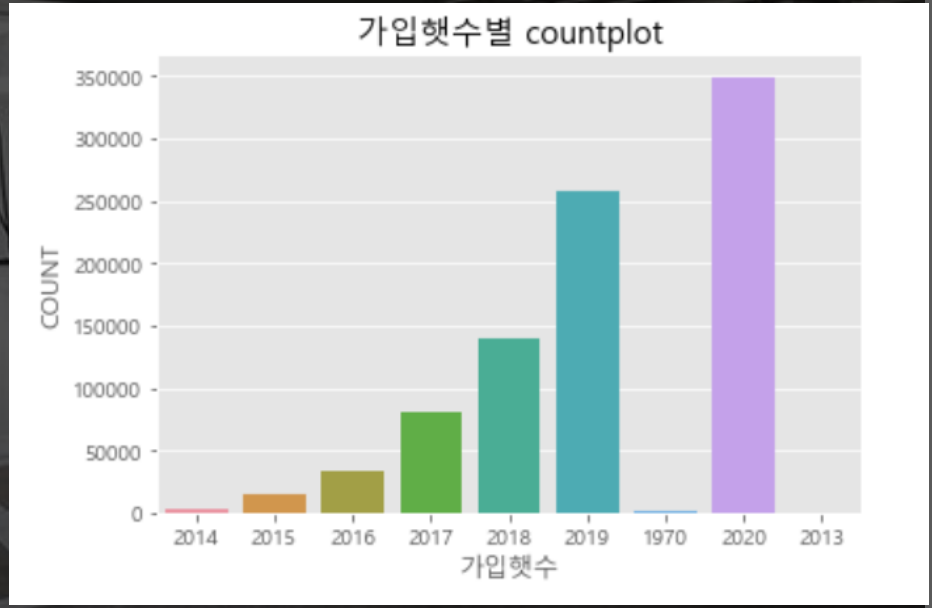
# Feature_engineering



## 데이터 기초 시각화

```
## 가입년도 별 countplot
sns.countplot(x='JOIN_DATE',data=df)
plt.xlabel('가입햇수')
plt.ylabel('COUNT')
plt.title('가입햇수별 countplot', fontsize=15)

plt.show()
```



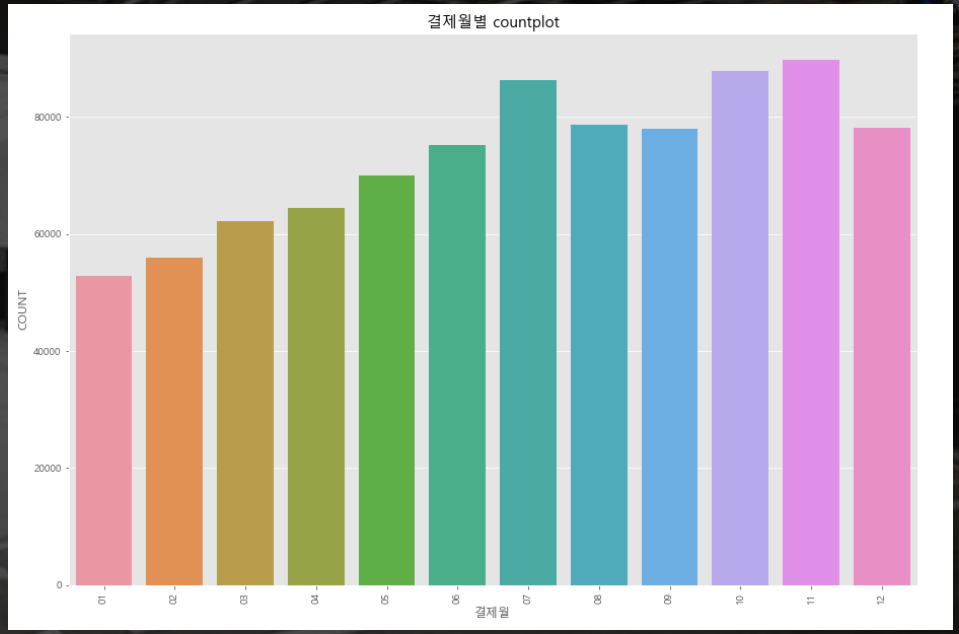

년도별 사용횟수가 늘어남
1970년대 이상치가 포함

# Feature_engineering

전처리 & EDA

## 데이터 기초 시각화

```
sns.countplot(x=df['MONTH'],data=df)
plt.xticks(rotation=90)
plt.xlabel('결제월')
plt.ylabel('COUNT')
plt.title('결제월별 countplot', fontsize=15)
plt.show()
```




월별 사용횟수가 증가하는 추세
혹서기로 생각되는 8~9월은 감소

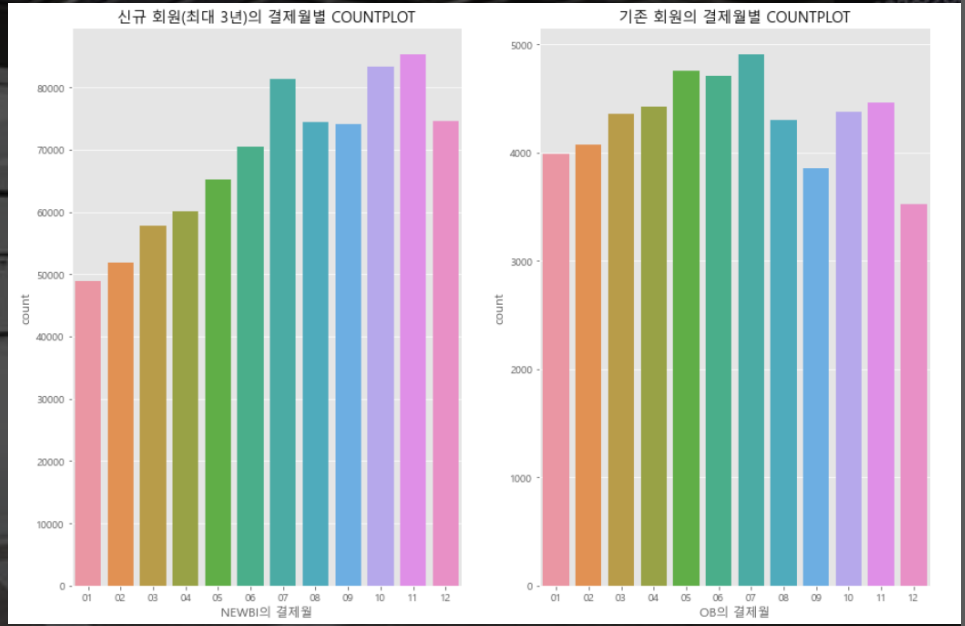
# Feature_engineering

전처리 & EDA

## 데이터 기초 시각화

```
# 신규 및 이전 가입자 COUNT 시각화
plt.rcParams['figure.figsize'] = [15,10]
plt.subplot(121)
sns.countplot(x=newbi['MONTH'], data=newbi)
plt.xlabel('NEWBI의 결제월')
plt.title('신규 회원(최대 3년)의 결제월별 COUNTPLOT')

plt.subplot(122)
sns.countplot(x=ob['MONTH'], data=ob)
plt.xlabel('OB의 결제월')
plt.title('기존 회원의 결제월별 COUNTPLOT')
plt.show()
```

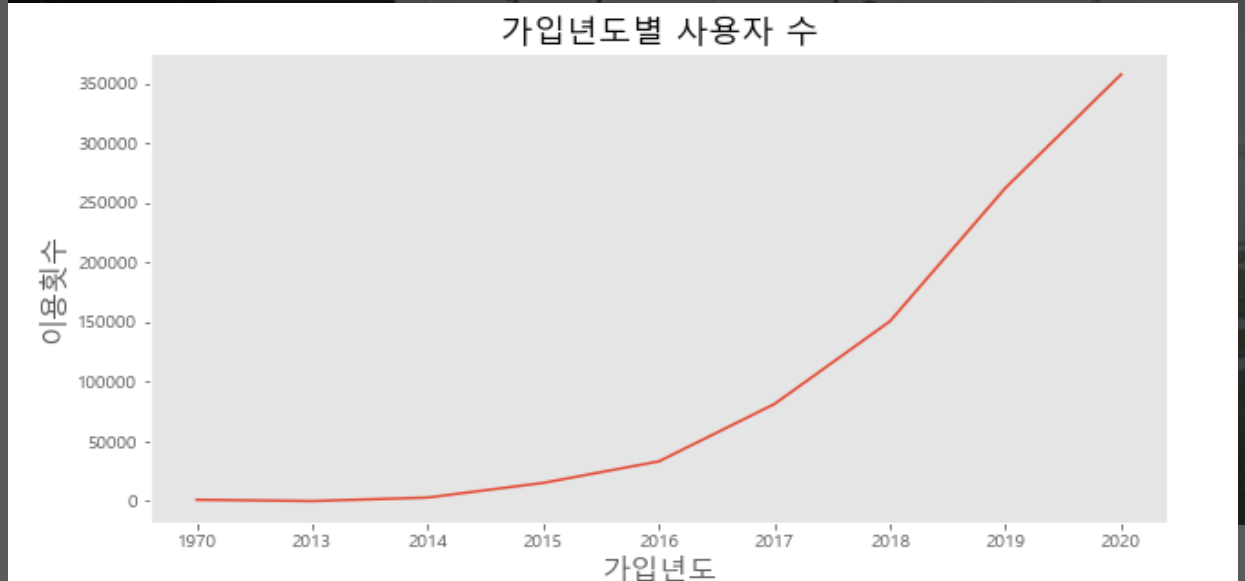



최근 3년 내 가입 회원과 기존 회원의 월별 소비 패턴은 차이를 보임
최근 가입한 사람의 데이터가 많음 (모집단과 동일한 패턴)

# Feature_engineering

## 전처리 & EDA

```
# 가입일자별로 이용횟수 파악
plt.figure(figsize=(10,5))
# x축 y축 설정
sns.lineplot(df.groupby("JOIN_DATE")["COUNT"].sum().keys(),df.groupby("JOIN_DATE")["COUNT"].sum()) # 2010년
plt.grid()
plt.title('가입년도별 사용자 수', fontsize=18)
plt.ylabel('이용횟수', fontsize=16)
plt.xlabel('가입년도', fontsize=16)
plt.show()
```




시간 흐름에 따른 이용횟수 증가

JOIN_DATE를
COUNT 예측에 활용

# Feature_engineering

전처리 & EDA

## 데이터 탐색

```
#USER_ID 컬럼 분포 확인
df_utype_count=df['USER_ID'].value_counts()
print(df_utype_count)
```

```
591610     244
1355841    235
1365069    222
1224426    222
1407916    222
          ...
586251       1
1575444      1
1573397      1
1591836      1
30735        1
Name: USER_ID, Length: 165424, dtype: int64
```

유저들의 사이의 사용 편차가 존재

유저별로 등급을 나눠
COUNT 예측 정확도를 높이자

# Feature_engineering



## 이상치 처리

**일반적인 이용자로 보이지 않는 데이터를 이상치로 판단**
**다음 조건에 해당하는 이용자는 이상치로 처리**

- 연속적으로 이용
- 데이터가 5개 이상
- AD1이 계속해서 바뀜
- COUNT 값의 평균이 비정상적으로 높음

```
# 이상치 파악
print("이상치 ID :", 999665)

# 이상치(회사에서 서비스를 파악하기 위해 사용되는 것 같은 ID) 제거
df = df[df["USER_ID"] != 999665]
df = df.reset_index()
df.drop("index", axis=1, inplace=True)
```

```
이상치 ID : 999665
```

- 이상치로 보이는 999665 ID값 제거

# Feature_engineering

## 전처리 & EDA

```
# 공휴일 컬럼 만들기

# 2020년 휴일 리스트
holiday_list = ['2020-01-01', '2020-01-24', '2020-01-25', '2020-01-26', '2020-01-27', '2020-03-01', '2020-05-05',

# 보다 빠른 계산을 위해 값 변환보다 list를 column으로 선언하는 방법 사용
day_type_list = []


for z in range(len(df)) :
    if df['DATE'].loc[z].strftime('%Y-%m-%d') in holiday_list :
        day_type_list.append("공휴일")

    elif df['DATE'].loc[z].weekday() > 4 :
        day_type_list.append("주말")

    else :
        day_type_list.append("주중")

df["DAY_TYPE"] = day_type_list


# 데이터 확인
df_day_0 = df[df["DAY_TYPE"]=="주말"]
df_day_1 = df[df["DAY_TYPE"]=="공휴일"]
df_day_2 = df[df["DAY_TYPE"]=="주중"]

print("전체데이터 중 주말의 비율 :", (len(df_day_0)/len(df))*100,"%")
print("전체데이터 중 주중의 비율 :", (len(df_day_1)/len(df))*100,"%")
print("전체데이터 중 공휴일의 비율 :", (len(df_day_2)/len(df))*100,"%")
print()
print("365일 중 주말의 비율 :",(df_day_0["DATE"].nunique()/365)*100, "%")
print("365일 중 주중의 비율 :",(df_day_1["DATE"].nunique()/365)*100, "%")
print("365일 중 공휴일의 비율 :", (df_day_2["DATE"].nunique()/365)*100, "%")
```

**요일별 정확한 분석을 위한**
**주중, 주말, 공휴일 구분**

# Feature_engineering

## 전처리 & EDA

전체데이터 중 주말의 비율 : 23.69789681478658 %
전체데이터 중 주중의 비율 : 2.8692520807140705 %
전체데이터 중 공휴일의 비율 : 73.43285110449935 %

365일 중 주말의 비율 : 26.84931506849315 %
365일 중 주중의 비율 : 4.657534246575342 %
365일 중 공휴일의 비율 : 68.76712328767123 %

주말 데이터입니다.

데이터 중 4보다 큰 COUNT 항의 개수 : 106
데이터 중 1보다 큰 COUNT 항의 개수 : 3721

전체 주말데이터의 개수 : 208282
데이터 중 5회 이상 사용한 사람의 비율 : 0.05089253992183674
데이터 중 2회 이상 사용한 사람의 비율 : 1.7865201985769292

공휴일 데이터입니다.

데이터 중 4보다 큰 COUNT 항의 개수 : 7
데이터 중 1보다 큰 COUNT 항의 개수 : 472

전체 주말데이터의 개수 : 25218
데이터 중 5회 이상 사용한 사람의 비율 : 0.027757950670156 24
데이터 중 2회 이상 사용한 사람의 비율 : 1.871678959473392

주중 데이터입니다.

데이터 중 4보다 큰 COUNT 항의 개수 : 167
데이터 중 1보다 큰 COUNT 항의 개수 : 10937

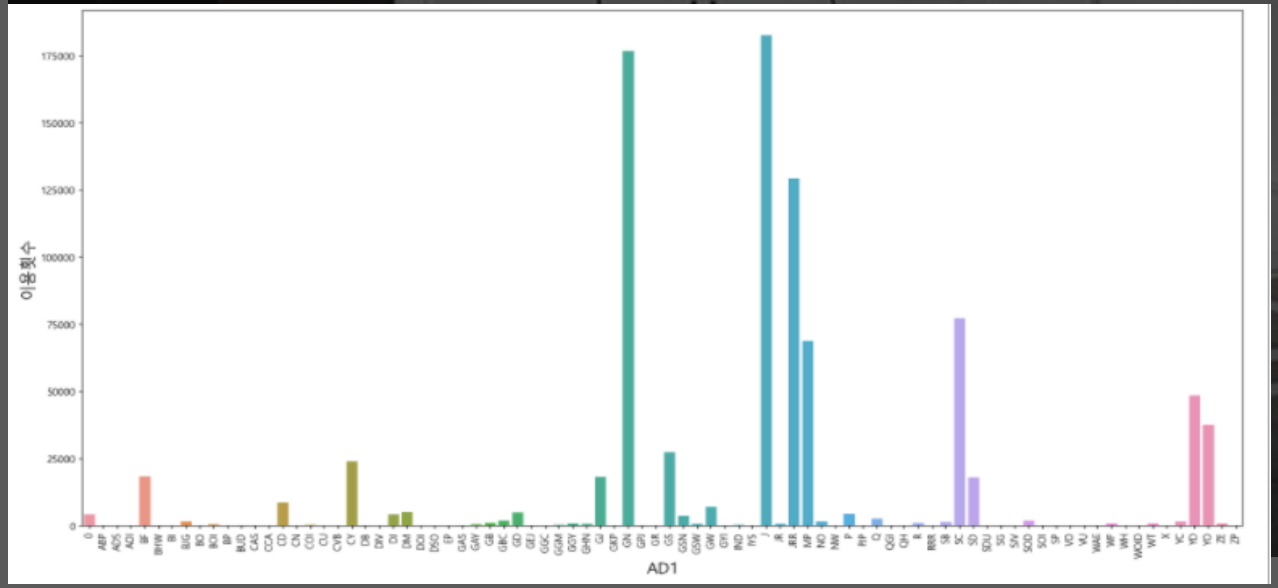
전체 주말데이터의 개수 : 645405
데이터 중 5회 이상 사용한 사람의 비율 : 0.025875225633516942
데이터 중 2회 이상 사용한 사람의 비율 : 1.6945948667890702

**주말에 이용하는 고객이 다소 자주 이용할 확률이 높음**

# Feature_engineering

## 전처리 & EDA

```
# AD1별 이용횟수별 시각화
plt.figure(figsize=(20,10))
sns.barplot(x=df.groupby("AD1")["COUNT"].sum().keys(),y=df.groupby("AD1")["COUNT"].sum())
plt.ylabel('이용횟수', fontsize=16)
plt.xlabel('AD1', fontsize=16)
plt.xticks(rotation='vertical')
plt.show()
```




AD1의 특정 변수가
자주 이용됨

# Feature_engineering

전처리 & EDA

## STORE_ID / AD1

```
#STORE_ID와 AD1과의 상관관계
ad1_list = df['AD1'].unique()
for k in ad1_list :
    new_df = df[df['AD1']==k]
    print(k,"의 STORE_ID 리스트 : ", new_df['STORE_ID'].unique())
```

```
GAY 의 STORE_ID 리스트 :  [110484 220746 203627 220808]
GYI 의 STORE_ID 리스트 :  [220794 220385]
CCA 의 STORE_ID 리스트 :  [220673 223626]
PJP 의 STORE_ID 리스트 :  [220802 220769 220767 220748 222827 220785 230952]
GGC 의 STORE_ID 리스트 :  [220807]
```

STORE_ID는 AD1을 구성
AD1이 STORE_ID의 속성을 포함

STORE_ID의 drop 필요성

# Feature_engineering



## AD1 이니셜 추출

```
seoul=['JR', 'J', 'YO', 'SOD', 'GJ', 'DM', 'JRR', 'SB', 'GB', 'DB', 'NW',
in_seoul = []
out_of_seoul = []
for x in df['AD1'].unique() :
    new_df = df[df['AD1']==x]
    if x in seoul :
        in_seoul.append(new_df['STORE_ID'].nunique())
    else :
        out_of_seoul.append(new_df['STORE_ID'].nunique())

print("서울 AD1 STORE_ID 평균 개수 :",sum(in_seoul)/25)
print("서울 외 AD1 STORE_ID 평균 개수 :",sum(out_of_seoul)/60)
```

```
서울 AD1 STORE_ID 평균 개수 : 28.64
서울 외 AD1 STORE_ID 평균 개수 : 5.7333333333333333
```

서울로 추정되는 이니셜 25개와
그 외 이니셜 STORE_ID 개수 차이

서울 내 주차장 수가 많음
해당 추측은 일치하는 것으로 판단

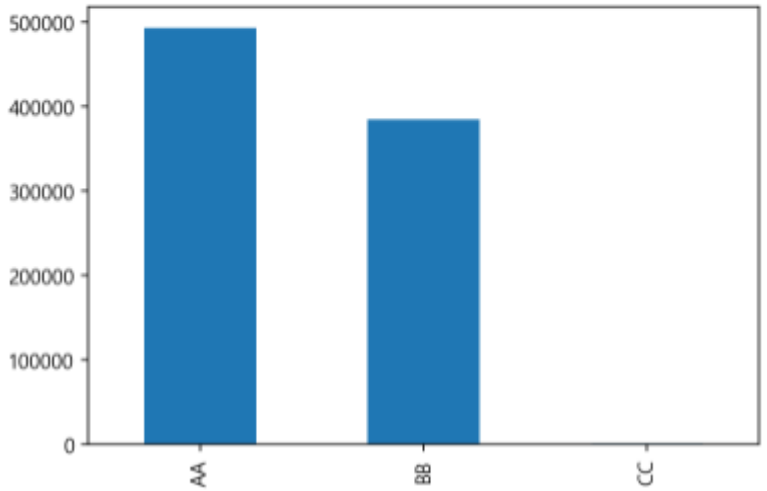
# Feature_engineering

## 전처리 & EDA

### 데이터 탐색

```
#D_TYPE 컬럼 분포 확인
df_dtype_count=df['D_TYPE'].value_counts()
print(df_dtype_count)
print(df_dtype_count.plot.bar())
```

```
AA    493166
BB    384541
CC      1198
Name: D_TYPE, dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
```




D_TYPE이 CC인 유저 수가
다른 두가지 D_TYPE에 비해
압도적으로 적음

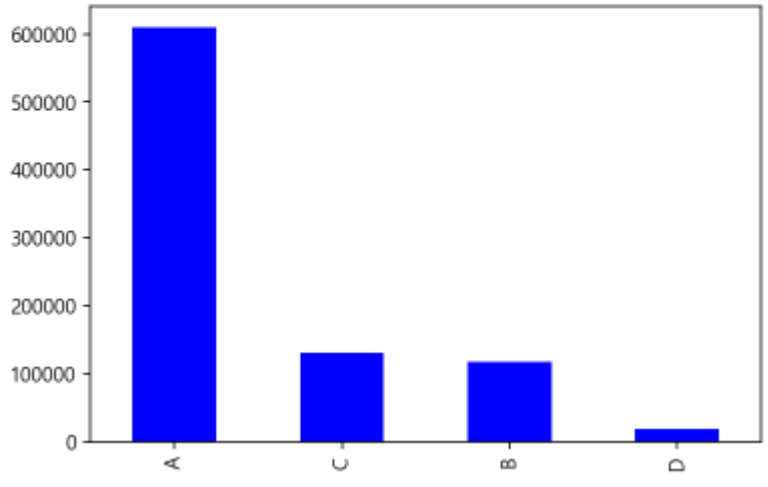
# Feature_engineering

전처리 & EDA

## 데이터 탐색

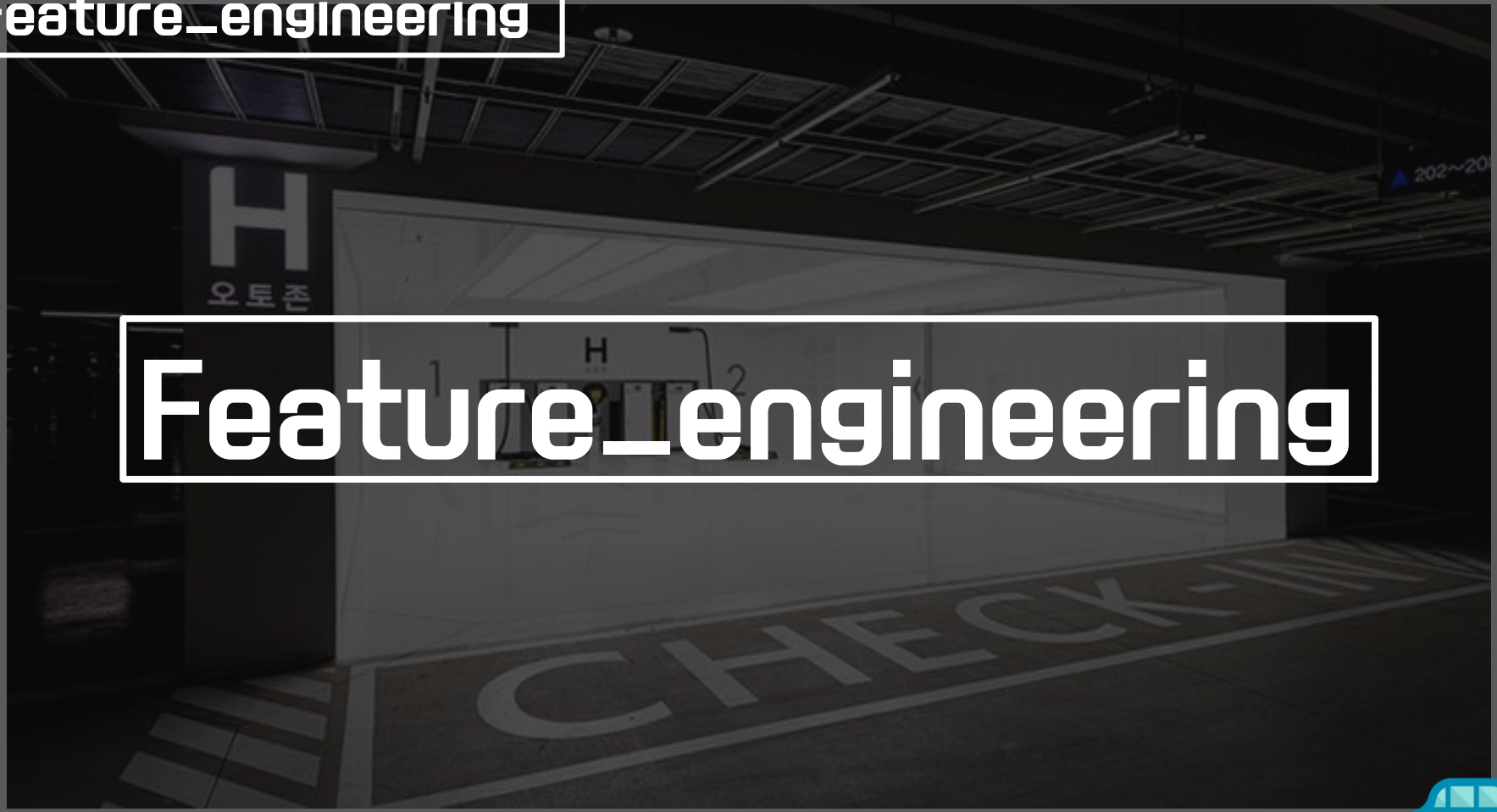
```
#GOODS_TYPE 컬럼 분포 확인
df_gtype_count=df['GOODS_TYPE'].value_counts()
print(df_gtype_count)
print(df_gtype_count.plot.bar(color='blue'))
```

```
A    609494
C    131108
B    118541
D     19762
Name: GOODS_TYPE, dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
```



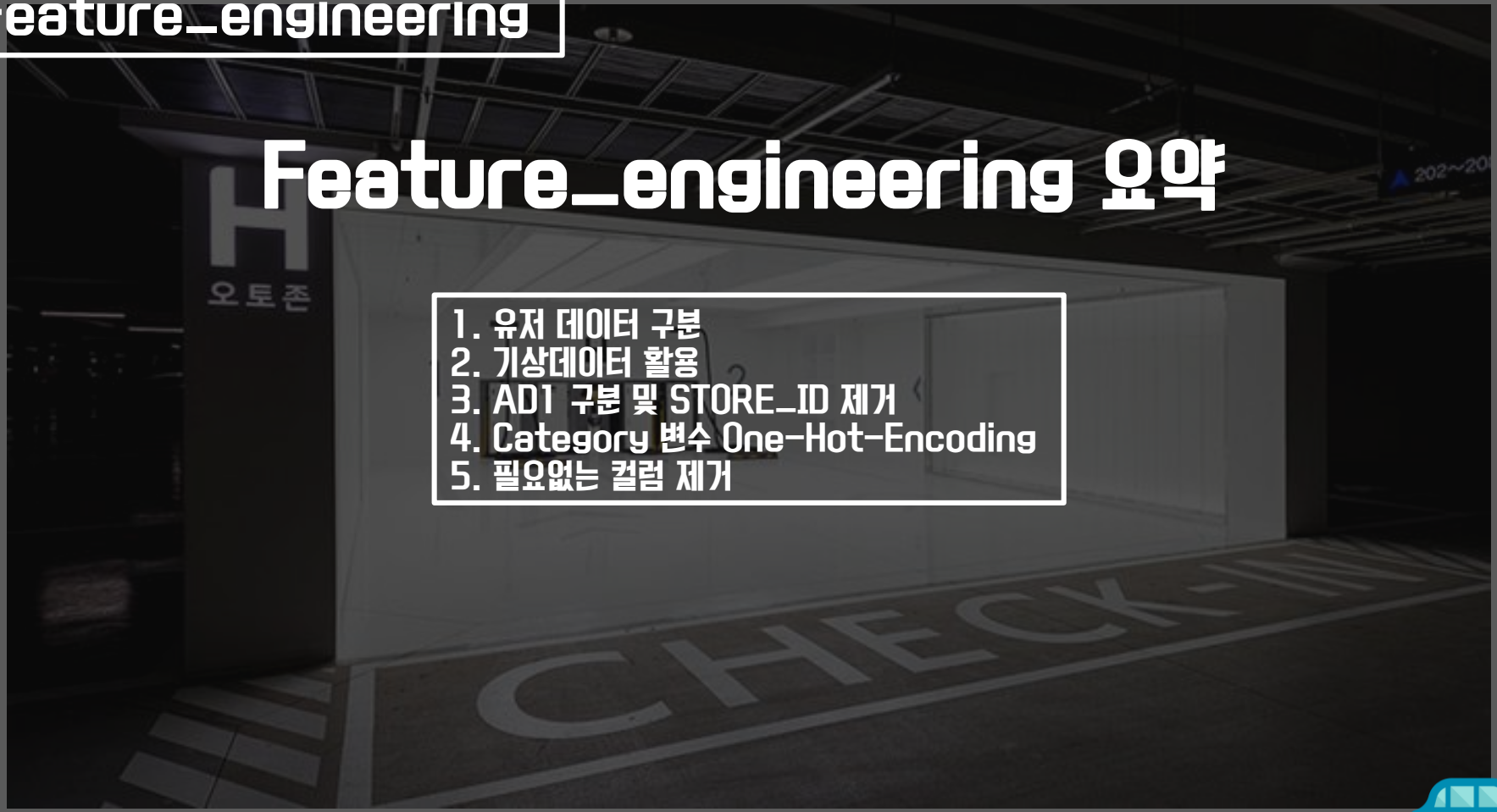

GOODS_TYPE은 A의 사용이
다른 타입보다 압도적으로 많음

Feature_engineering




# Feature_engineering

Feature_engineering




# Feature_engineering 요약

1. 유저 데이터 구분
2. 기상데이터 활용
3. AD1 구분 및 STORE_ID 제거
4. Category 변수 One-Hot-Encoding
5. 필요없는 컬럼 제거

# Feature_engineering

Feature_engineering

## 유저 데이터 구분

```
df_ucm_1=df_ucm[df_ucm['MEAN_COUNT']==1]
df_ucm_1['USER_ID_TYPE']='A'
df_ucm_1=df_ucm_1[['USER_ID','USER_ID_TYPE']]
df_ucm_2=df_ucm[(df_ucm['MEAN_COUNT']>=3) & (df_ucm['COUNTS']>=2)]
df_ucm_2['USER_ID_TYPE']='B'
df_ucm_2=df_ucm_2[['USER_ID','USER_ID_TYPE']]
df_ucm_3=df_ucm[df_ucm['MEAN_COUNT']!=1]
df_ucm_3=df_ucm_3[(df_ucm_3['MEAN_COUNT']<3) | (df_ucm_3['COUNTS']==1)]
df_ucm_3['USER_ID_TYPE']='C'
df_ucm_3=df_ucm_3[['USER_ID','USER_ID_TYPE']]
df_ucm_3
df_ut=pd.concat([df_ucm_1,df_ucm_2,df_ucm_3])
```

## A, B, C 등급으로 구분

# Feature_engineering

Feature_engineering

## 기상 데이터 활용

```
# 폭염과 한파
temperatures = df_w[(df_w['기온'] > 33) | (df_w['기온'] < -12)]
day = list(temperatures['일자'].unique())
# 이상 기온 여부 구분
everyday['이상기온'] =0
for i in range(len(everyday)):
    if everyday['일자'].iloc[i] in day:
        everyday['이상기온'].iloc[i] = 1
# 최종 강수여부, 이상기온 여부 데이터프레임
weather_col=everyday[['일자', '강수여부','이상기온']]
# 마지막 누락된 데이터 추가
weather_col.loc[365] = ['2020-12-31 00:00:00', 0, 1]
weather_col['일자'] = pd.to_datetime(weather_col['일자'])
```

## 강수여부, 이상기온 여부 판단

| | 일자 | 강수여부 | 이상기온 |
|---|---|---|---|
| 0 | 2020-01-01 | 1 | 0 |
| 1 | 2020-01-02 | 0 | 0 |
| 2 | 2020-01-03 | 0 | 0 |
| 3 | 2020-01-04 | 0 | 0 |
| 4 | 2020-01-05 | 0 | 0 |
| ... | ... | ... | ... |
| 361 | 2020-12-27 | 0 | 0 |
| 362 | 2020-12-28 | 0 | 0 |
| 363 | 2020-12-29 | 0 | 0 |
| 364 | 2020-12-30 | 0 | 1 |
| 365 | 2020-12-31 | 0 | 1 |

# Feature_engineering

Feature_engineering

## AD1 구분 및 STORE_ID 제거

```
# 서울 내외 구분

# 컬럼 생성용 리스트 선언
ad1_type_list = []

# 빠른 계산을 위한 list
for k in range(len(df_day)) :
    # AD1_TYPE 파악
    if df_day["AD1"].loc[k] in seoul :
        ad1_type_list.append("SEOUL")
    else :
        ad1_type_list.append("NOT_SEOUL")

# 서울 구분 컬럼 생성
df_day['AD1_TYPE'] = ad1_type_list
```

```
# STORE_ID 제거하기
df_day = df_day.drop("STORE_ID", axis=1)
```

동일한 설명력을 갖는 컬럼 제거
다중공선성 제거

# Feature_engineering

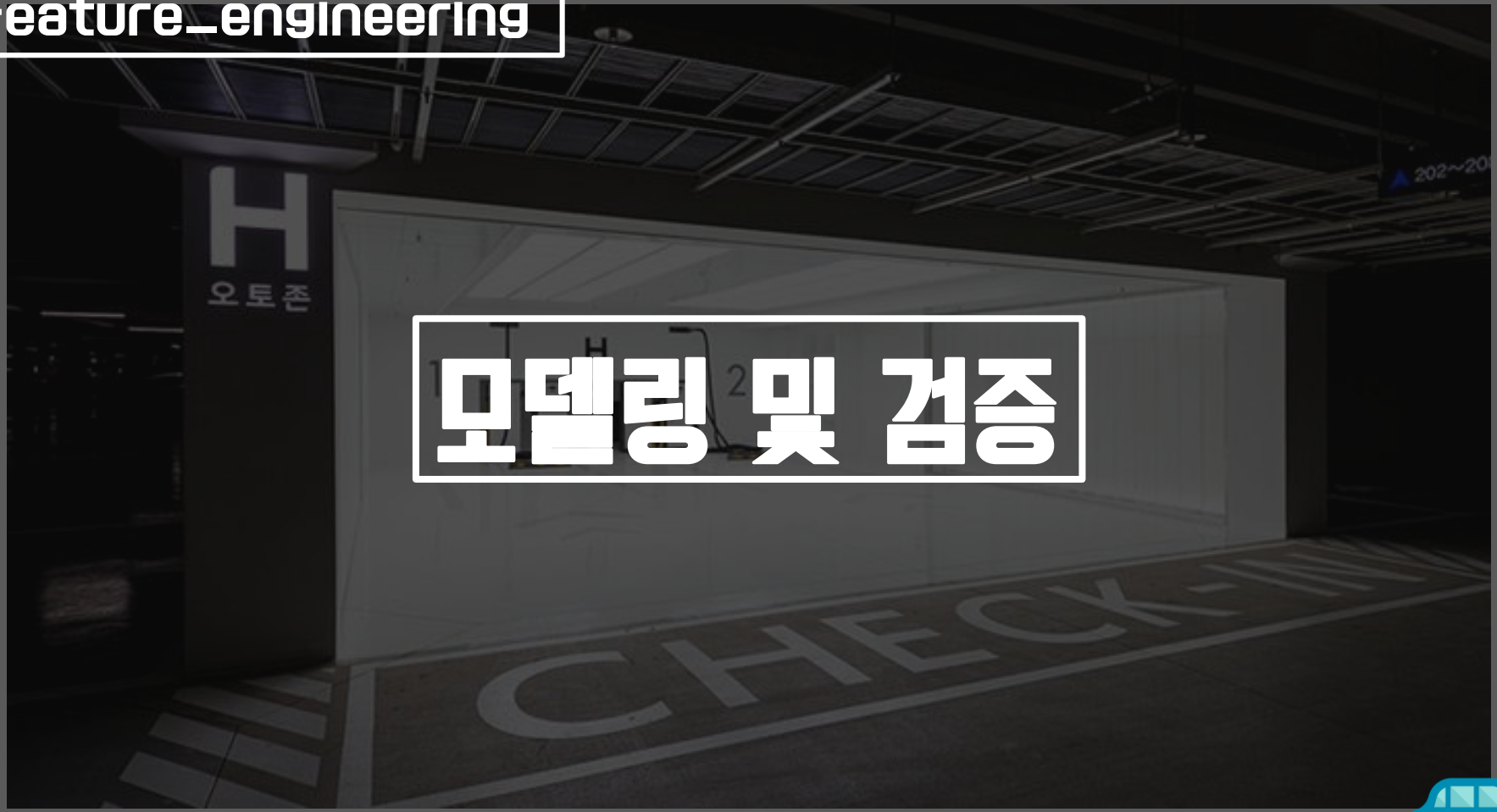
Feature_engineering

## Category 변수 One-Hot-Encoding
## 필요없는 컬럼 제거

```
# 카테고리값을 One_hot_encoding
df_day = pd.get_dummies(df_day, columns=["JOIN_DATE", "DAY_TYPE", "AD1_TYPE"])
```

```
# 필요없는 컬럼 drop
df_day = df_day.drop(["AD1","AD1_TYPE_NOT_SEOUL"], axis=1)
```

Feature_engineering




# 모델링 및 검증

Feature_engineering

# 모델링 및 검증 요약

1. 일일 COUNT 예측
   - 훈련 및 테스트 데이터 설정
   - 최선의 모델을 찾기 위한 다양한 모델 설정
   - 선정된 모델의 최적 하이퍼 파라미터 찾기
   - MSE 값 측정
   - Feature Importance 확인

2. 3달 COUNT 예측
   - 훈련 및 테스트 데이터 설정
   - Min-Max scaling
   - 최선의 모델을 찾기 위한 다양한 모델 설정
   - 선정된 모델의 최적 하이퍼 파라미터 찾기
   - MSE, MAE 값 측정
   - Feature Importance 확인

# Feature_engineering

일일 COUNT 예측

## 데이터 나누기 및 필요없는 컬럼 제거

```
# 시간순서로 나열
df_day1= df_day1.sort_values(by="DATE")

# 데이터 나누기
X_train1 = df_day1[:623305]
X_test1 = df_day1[623305:]

# 필요없는 컬럼 제거
train_x1 = X_train1.drop(["DATE", "COUNT"], axis=1)
train_y1 = X_train1['COUNT']

test_x1 = X_test1.drop(["DATE", "COUNT"], axis=1)
real_count1 = X_test1['COUNT']
```

D_TYPE, GOODS_TYPE, JOIN_DATE 등
일일 예측에 관련 없는 데이터 제거

1~9월 데이터 : 훈련데이터 설정
10~12월 데이터 : 테스트데이터 설정

Count를 타겟데이터로 설정

# Feature_engineering

## 일일 COUNT 예측

### 최선의 모델 찾기
### MSE score 출력 함수 설정

```
# 각 모델별 MSE 값 확인 (디폴트 설정)
np.random.seed(42)

gboost = GradientBoostingRegressor()
xgboost = XGBRegressor()
lightgbm = LGBMRegressor()
rdforest = RandomForestRegressor()

models1 = [gboost, xgboost, lightgbm, rdforest]
```

```
def get_scores(models1, train_x1, train_y1, test_x1):
    df2 = {}

    for model in models1:
        model.fit(train_x1, train_y1)
        y_pred1 = model.predict(test_x1)
        print("사용한 모델 :",model)
        print("MSE :",mean_squared_error(real_count1, y_pred1))
        print()
        print()
```

```
사용한 모델 : GradientBoostingRegressor()
MSE : 0.03745320059513653


사용한 모델 : XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
MSE : 0.038953190867117314


사용한 모델 : LGBMRegressor()
MSE : 0.036218743189760254


사용한 모델 : RandomForestRegressor()
MSE : 0.04461078021225606
```

**LightGBM 선정**

# Feature_engineering

## 일일 COUNT 예측

### 최적 모델의 최적 하이퍼파라미터 찾기

```python
# lightgbm이 가장 낮음
# 해당 모델의 가장 좋은 파라미터를 찾기

# 파라미터 범위
param_grid = {
    'n_estimators': [10, 30, 50],
    'max_depth': [10, 20],
    'learning_rate' : [0.05],
    'num_iterations' : [100,200],
}

# 모델 준비 (LGBMRegressor)
model1 = LGBMRegressor()
```

```python
# GridSearch로 가장 좋은 파라미터 찾기
GCV = GridSearchCV(model1, param_grid=param_grid, scoring='neg_mean_squared_error',

# 모델 fitting
GCV.fit(train_x1, train_y1)

# 가장 좋은 파라미터 찾기
print("Best Param :", GCV.best_params_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:  1.4min
[Parallel(n_jobs=5)]: Done  60 out of  60 | elapsed:  2.1min finished

Best Param : {'learning_rate': 0.05, 'max_depth': 10, 'n_estimators': 10, 'num_iterations': 100}
```

# Feature_engineering

## 입입 COUNT 예측

### 최적 하이퍼파라미터 설정 후 예측

```
# 가장 좋은 파라미터로 모델 설정
model1=GCV.best_estimator_

# 예측
y_pred1 = model1.predict(test_x1)
```

```
print("MSE :",mean_squared_error(real_count1, y_pred1))
```

```
MSE : 0.03560501243087223
```

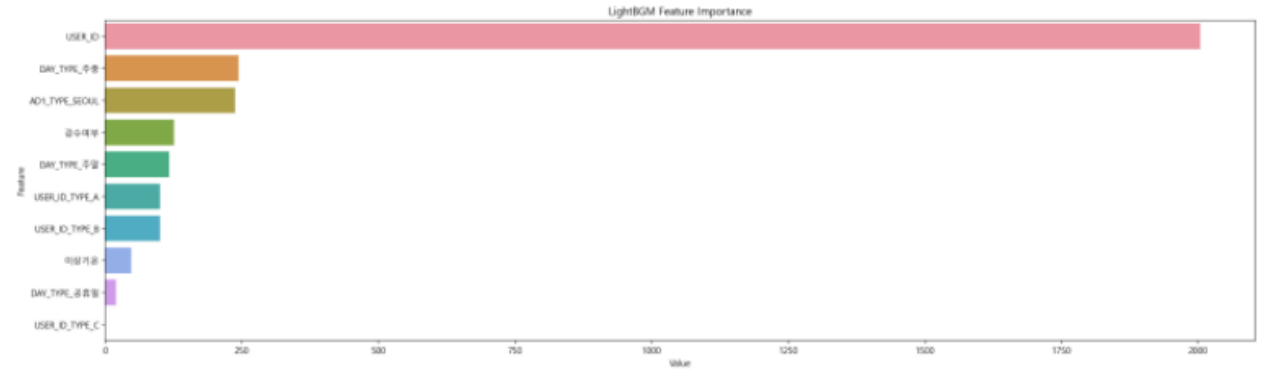
# Feature_engineering

## 일일 COUNT 예측

## Feature Importance 확인

```
# Feature importance
feature_imp = pd.DataFrame(sorted(zip(model1.feature_importances_,train_x1.columns)), columns=['Value','Featur

plt.figure(figsize=(20, 6))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False).head(20))
plt.title('LightBGM Feature Importance')
plt.tight_layout()
plt.show()
```



# Feature_engineering

## 월별 COUNT 예측

### 데이터 나누기 및 필요없는 컬럼 제거

```
#날짜기준 train data set, test data set 분리
df_final.sort_values(by='DATE',ascending=True,inplace=True)
df_final.reset_index(inplace=True)
df_final.drop(['index'],axis=1,inplace=True)
df_train_tem=df_final.loc[:623304]
df_test_tem=df_final.loc[623305:]
```

D_TYPE, GOODS_TYPE, JOIN_DATE 등
월별 예측에 필요한 데이터 포함

1~9월 데이터 : 훈련데이터 설정
10~12월 데이터 : 테스트데이터 설정

Count를 타겟데이터로 설정

# Feature_engineering

## 월별 COUNT 예측

### Min-Max Scaling

```
#min-max 정규화 함수
def mmnorm(s):
    return (s - s.min()) / ( s.max() - s.min())
```

```
#train data set 전처리
#정규화
df_train['공휴일']=mmnorm(df_train[['공휴일']])
df_train['주말']=mmnorm(df_train[['주말']])
df_train['주중']=mmnorm(df_train[['주중']])
df_train['강수여부']=mmnorm(df_train[['강수여부']])
df_train['이상기온']=mmnorm(df_train[['이상기온']])
df_train['D_AA']=mmnorm(df_train[['D_AA']])
df_train['D_BB']=mmnorm(df_train[['D_BB']])
df_train['D_CC']=mmnorm(df_train[['D_CC']])
df_train['GOODS_A']=mmnorm(df_train[['GOODS_A']])
df_train['GOODS_B']=mmnorm(df_train[['GOODS_B']])
df_train['GOODS_C']=mmnorm(df_train[['GOODS_C']])
df_train['GOODS_D']=mmnorm(df_train[['GOODS_D']])
```

합계로 인해 커진 변수값
min-max 함수로 scaling

# Feature_engineering

## 원별 COUNT 예측

최선의 모델 찾기
MSE score 출력
MAE score 출력

```
#랜덤포레스트
rf = RandomForestRegressor()
rf.fit(df_train_x,df_train_y)

#gboost
gb=GradientBoostingRegressor()
gb.fit(df_train_x,df_train_y)

#lightgbm
lg=LGBMRegressor()
lg.fit(df_train_x,df_train_y)
```

```
LGBMRegressor()
```

```
rf_y_pred=rf.predict(df_test_x)
print("rf mse :",mean_squared_error(df_test_y,rf_y_pred))
print("rf mae :",mean_absolute_error(df_test_y,rf_y_pred))
gb_y_pred=gb.predict(df_test_x)
print("gb mse :",mean_squared_error(df_test_y,gb_y_pred))
print("gb mae :",mean_absolute_error(df_test_y,gb_y_pred))
lg_y_pred=lg.predict(df_test_x)
print("lg mse :",mean_squared_error(df_test_y,lg_y_pred))
print("lg mae :",mean_absolute_error(df_test_y,lg_y_pred))
```

```
rf mse : 0.3719841068114661
rf mae : 0.10656131371471773
gb mse : 0.2726050247630921
gb mae : 0.12493188245452637
lg mse : 0.26389459776750673
lg mae : 0.10399234125248175
```

LightGBM 선정

# Feature_engineering

## 월별 COUNT 예측

### 최적 모델의 최적 하이퍼파라미터 찾기

```
# lightgbm이 가장 낮음
# 해당 모델의 가장 좋은 파라미터를 찾기

# 파라미터 범위
param_grid = {
    'n_estimators': [10, 30, 50],
    'max_depth': [10, 20],
    'learning_rate' : [0.05],
    'num_iterations' : [100,200],
}

# 모델 준비 (LGBMRegressor)
model1 = LGBMRegressor()
```

```
# GridSearch로 가장 좋은 파라미터 찾기
GCV = GridSearchCV(lg_model, param_grid=param_grid, scoring='neg_mean_squared_error',
# 모델 fitting
GCV.fit(df_train_x, df_train_y)
# 가장 좋은 파라미터 찾기
print("Best Param :", GCV.best_params_)
# 가장 좋은 파라미터로 모델 설정
lg_model=GCV.best_estimator_
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:  1.0min
[Parallel(n_jobs=5)]: Done  60 out of  60 | elapsed:  1.5min finished
Best Param : {'learning_rate': 0.05, 'max_depth': 20, 'n_estimators': 10, 'num_iterations': 200}
```

# Feature_engineering

## 월별 COUNT 예측

### 최적 하이퍼파라미터 설정 후 예측

```
#예측
lg_y_pred=lg_model.predict(df_test_x)
print('최종 mse(lightgbm모델 사용) : ',mean_squared_error(df_test_y,lg_y_pred))
print('최종 mae(lightgbm모델 사용) : ',mean_absolute_error(df_test_y,lg_y_pred))
```

```
최종 mse(lightgbm모델 사용) :  0.26355187894719134
최종 mae(lightgbm모델 사용) :  0.10392207522470769
```

# Feature_engineering

## 월별 COUNT 예측

## Feature Importance 확인

```
# Feature importance
feature_imp = pd.DataFrame(sorted(zip(lg_model.feature_importances_,df_train_x.columns)), columns=['Value','Fe

plt.figure(figsize=(20, 6))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", ascending=False).head(20))
plt.title('LightBGM Feature Importance')
plt.tight_layout()
plt.show()
```