

## 날씨/대기오염에 따른 서울시 공공 자전거 이용자 수 예측



따릉2조

김지현 윤다영 이다빈 지현진

실시간 빅데이터 분석 및 시각 인지 시스템 개발자 양성과정

# Content

1. 프로젝트 개요 및 목적
2. 연구 모형
3. 데이터 수집 및 전처리
4. EDA
5. 회귀분석
6. 결론

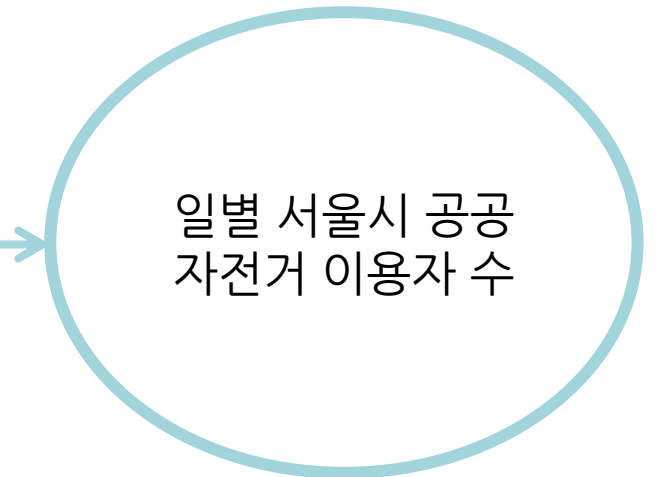
개요	내용
프로젝트명	날씨/대기오염에 따른 서울시 공공 자전거 이용자 수 예측
개발자	김지현 윤다영 이다빈 지현진
사용언어	Python
개발환경 및 라이브러리	Colab / Numpy, Pandas, Matplotlib, Sklearn, Seaborn, ...
프로젝트 개요	날씨의 형태(유형별), 대기오염 등 공기의 질에 따른 분석을 바탕으로 날씨와 공기의 질이 따름이 이용객들에게 미치는 영향에 관한 연구 모델을 제시하고자 한다.

프로젝트 일정	일정					
	9/30	10/1	10/5	10/6	10/7	10/8
주제 선정						
데이터 수집 및 전처리						
EDA						
분석						
발표 준비						

## Input Variable





## Target Variable



Input, Target Variables = Numerical Variables



서울시가 공공자전거 ‘따릉이’를 본격적으로 도입한 시기는 2000년 4월으로 매년 그 이용자 수가 증가 하고 있다. 2017년 그 수가 급격하게 늘었으며, 2021년 현재 서울 시민 3명 중 1 명이 ‘따릉이’를 탄다고 할 수 있을 만큼 이용자 수가 많이 증가 하였다. 하지만 따릉이 수는 한정적이며, 서울시에 따르면 따릉이 사업이 적자사업인 만큼 당분간은 따릉이 대여소나 수를 늘릴 계획이 없는 입장을 밝힌 상태에서 최대한 이용자들의 수요를 충족하기 위해 계절, 날씨 등의 환경적 요인이 따릉이 이용에 어떠한 영향을 미치는 지 확인하고 이런 이용자 수 예측모델이 활용하여 효율적인 따릉이 배치를 하고자 한다.

개요	날씨 데이터	대기오염 데이터	따릉이 데이터
기간	2017 ~ 2020년		
위치	한강공원이 위치한 총 8개 서울시 구 -> 영등포구, 마포구, 서초구, 송파구, 광진구, 용산구, 강동구, 강서구		
자료형태	일일 데이터		
출처	 기상청 기상자료개방포털	 서울 열린데이터 광장	 서울 열린데이터 광장
수집데이터	평균기온, 강수량, 평균풍속	이산화질소, 이산화탄소, 아황산가스, 미세먼지, 초미세먼지, 오존	따릉이 대여/반납 건

날씨데이터	대기데이터
지점	이산화질농도
지점명	오존농도
일시	이산화탄소농도
평균기온	아황산가스
일강수량	미세먼지
평균 풍속	초미세먼지
일시	일시

마포 -> 용산, 광진 -> 성동, 강동 -> 송파, 서초 -> 강남, 영등포 -> 강서

```
def filldata(df, col):
    for i in range(len(df)-1):
        i += 1
        if math.isnan(df[col][i]):
            if df['지점명'][i] == '서초구':
                df[col][i] = df.loc[(df['지점명'] == '강남구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '마포구':
                df[col][i] = df.loc[(df['지점명'] == '용산구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '용산구':
                df[col][i] = df.loc[(df['지점명'] == '마포구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '광진구':
                df[col][i] = df.loc[(df['지점명'] == '송파구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '강동구':
                df[col][i] = df.loc[(df['지점명'] == '송파구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '송파구':
                df[col][i] = df.loc[(df['지점명'] == '강남구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '영등포구':
                df[col][i] = df.loc[(df['지점명'] == '강서구') & (df['일시'] == df['일시'][i]), [col]].values
            elif df['지점명'][i] == '강서구':
                df[col][i] = df.loc[(df['지점명'] == '영등포구') & (df['일시'] == df['일시'][i]), [col]].values
            else:
                continue
        else:
            continue
    return df
```

```
import math
filldata(df, '평균기온(° C)')
filldata(df, '일강수량(mm)')
filldata(df, '평균 풍속(m/s)')
```

따릉이 대여소 정보	따릉이 대여 정보
대여소 번호	대여소번호
보관소명	대여일시
자치구	대여소명
상세주소	대여거치대
위도	반납일시
경도	반납대여소번호
설치시기	반납대여소명
LCD	반납거치대
QR	이용시간
운영방식	이동거리



선택변수
대여소번호
자치구
대여일시



```
## 대여일자 & 지차구 기준 대여횟수 추출 (추후 merge위해 컬럼명 통일)  
df_group = df_drop.groupby(['대여일자', '지차구']).count()  
df_group.reset_index(inplace=True)  
df_group.columns = ['측정일자', '지점명', '대여횟수']  
len(df_group)
```

32686

대여일자와 지차구를 기준  
따릉이의 대여 횟수를 카운트 하여 타겟 변수를 생성



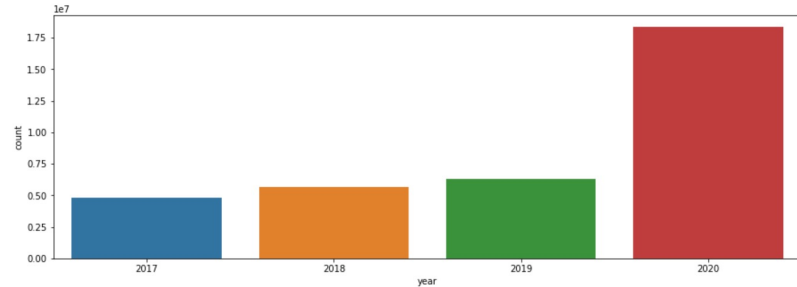
Dataset
측정일자
지점명
평균기온
일강수량
평균풍속
이산화질소농도
오존농도
이산화탄소농도
아황산가스
미세먼지
초미세먼지
대여횟수

## 04 EDA

### 데이터의 Target Variable 탐색 : Countplot을 사용해 데이터 확인

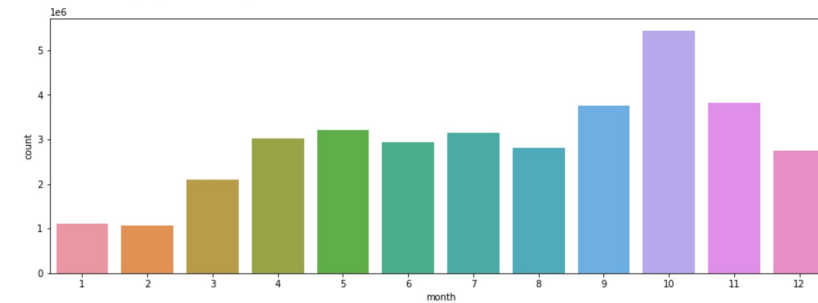
```
plt.figure(figsize=(15,5))
sns.countplot(data=df, x='year')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fdf8c5b7150>



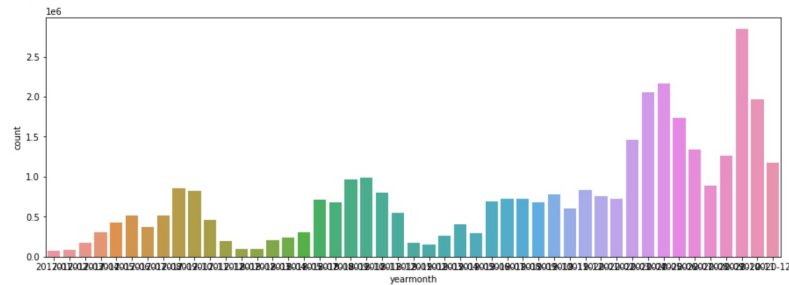
```
plt.figure(figsize=(15,5))
sns.countplot(data=df, x='month')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fdf8c1e9f10>



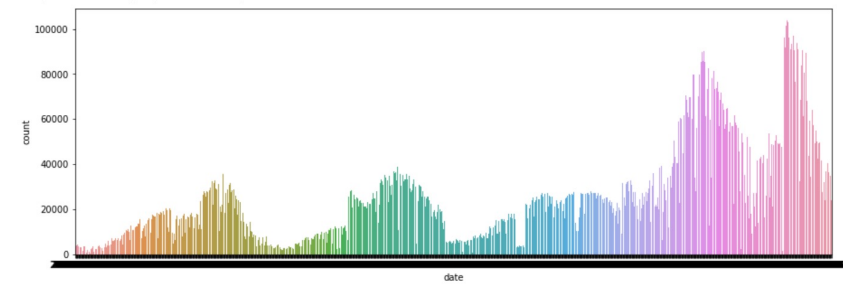
```
[12] plt.figure(figsize=(15,5))
sns.countplot(data=df, x='yearmonth')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7faccf964850>



```
plt.figure(figsize=(15,5))
sns.countplot(data=df, x='date')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fdf8c40d410>



공공 자전거 사용자 수가 계절 및 날씨의 영향을 받는다는 것을 확인했으며, 2020년 공공 자전거 수가 증가하면서 이용자 수가 늘어난 것이 확인

Metric : **R2 score** (상관관계가 높을수록 1에 가까워 짐)

### 선형 회귀모델

#### 1. Linear Regression

```
[16] from sklearn.metrics import r2_score, mean_squared_error
      from sklearn.metrics import r2_score
      linear_model = LinearRegression()
      linear_model.fit(X_train, y_train)
      y_predict = linear_model.predict(X_test)

      print("테스트 세트 R^2: {:.4f}".format(linear_model.score(X_test, y_test)))
```

테스트 세트 R^2: 0.1238

#### 2. Ridge Regression

```
[17] from sklearn.metrics import r2_score, mean_squared_error
      from sklearn.metrics import r2_score
      linear_model = Ridge()
      linear_model.fit(X_train, y_train)
      y_predict = linear_model.predict(X_test)

      print("테스트 세트 R^2: {:.4f}".format(linear_model.score(X_test, y_test)))
```

테스트 세트 R^2: 0.1236

Metric : **R2 score** (상관관계가 높을수록 1에 가까워 짐)

### 선형 회귀모델

#### 3. Lasso Regression

```
▶ from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import r2_score
linear_model = linear_model.Lasso()
linear_model.fit(X_train, y_train)
y_predict = linear_model.predict(X_test)

print("테스트 세트 R^2: {:.4f}".format(linear_model.score(X_test, y_test)))
```

테스트 세트 R^2: -0.0001

#### 4. Elastic Net Regression

```
[21] from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import r2_score
linear_model = ElasticNet()
linear_model.fit(X_train, y_train)
y_predict = linear_model.predict(X_test)

print("테스트 세트 R^2: {:.4f}".format(linear_model.score(X_test, y_test)))
```

테스트 세트 R^2: -0.0001

Metric : **R2 score** (상관관계가 높을수록 1에 가까워 짐)

### 비선형 회귀모델

#### 1. DecisionTreeRegressor

```
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import r2_score
linear_model = DecisionTreeRegressor()
linear_model.fit(X_train, y_train)
y_predict = linear_model.predict(X_test)

print("테스트 세트 R^2: {:.4f}".format(linear_model.score(X_test, y_test)))
```

테스트 세트 R^2: -0.2614



#### 2. K-Nearest Neighbors

```
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import r2_score
linear_model = KNeighborsRegressor()
linear_model.fit(X_train, y_train)
y_predict = linear_model.predict(X_test)

print("테스트 세트 R^2: {:.4f}".format(linear_model.score(X_test, y_test)))
```

테스트 세트 R^2: 0.2215

Metric : **R2 score** (상관관계가 높을수록 1에 가까워 짐)

```
[ ] import matplotlib.pyplot as plot
import numpy as np
import math
import pandas as pd
import seaborn as sns
import openpyxl
plot.rcParams["font.family"] = 'Malgun gothic'

from sklearn.neighbors import KNeighborsRegressor

for i in (20, 30, 40, 50):
    for j in ('uniform', 'distance'):
        for k in ('auto', 'ball_tree', 'kd_tree', 'brute'):
            model = KNeighborsRegressor(n_neighbors=i, weights=j, algorithm=k)
            model.fit(X_train, y_train)
            relation_square = model.score(X_test, y_test)
            print(f'결정계수 R{i}/{j}/{k}:', relation_square)
        print('###')
    print('\n')
```

파라미터 설정

결정계수 R20/uniform/auto: 0.28769877840252456	결정계수 R40/uniform/auto: 0.27206489130498623
결정계수 R20/uniform/ball_tree: 0.2877047032149218	결정계수 R40/uniform/ball_tree: 0.2720694522180518
결정계수 R20/uniform/kd_tree: 0.28769877840252456	결정계수 R40/uniform/kd_tree: 0.27206489130498623
결정계수 R20/uniform/brute: 0.28769877840252456	결정계수 R40/uniform/brute: 0.27207837608149454
결정계수 R20/distance/auto: 0.2969646763490488	결정계수 R40/distance/auto: 0.283243073950985
결정계수 R20/distance/ball_tree: 0.29697002863788924	결정계수 R40/distance/ball_tree: 0.2832460965440904
결정계수 R20/distance/kd_tree: 0.2969646763490488	결정계수 R40/distance/kd_tree: 0.283243073950985
결정계수 R20/distance/brute: 0.2969646763490488	결정계수 R40/distance/brute: 0.28325470372051564
결정계수 R30/uniform/auto: 0.2848718746738542	결정계수 R50/uniform/auto: 0.2619912847967377
결정계수 R30/uniform/ball_tree: 0.28488123398820586	결정계수 R50/uniform/ball_tree: 0.26199526671440454
결정계수 R30/uniform/kd_tree: 0.2848718746738542	결정계수 R50/uniform/kd_tree: 0.2619912847967377
결정계수 R30/uniform/brute: 0.2848718746738542	결정계수 R50/uniform/brute: 0.26199094421481184
결정계수 R30/distance/auto: 0.29445913714191263	결정계수 R50/distance/auto: 0.2742058936010411
결정계수 R30/distance/ball_tree: 0.29446759457938376	결정계수 R50/distance/ball_tree: 0.2742096271014084
결정계수 R30/distance/kd_tree: 0.29445913714191263	결정계수 R50/distance/kd_tree: 0.2742058936010411
결정계수 R30/distance/brute: 0.2944591371419125	결정계수 R50/distance/brute: 0.27420553838739026

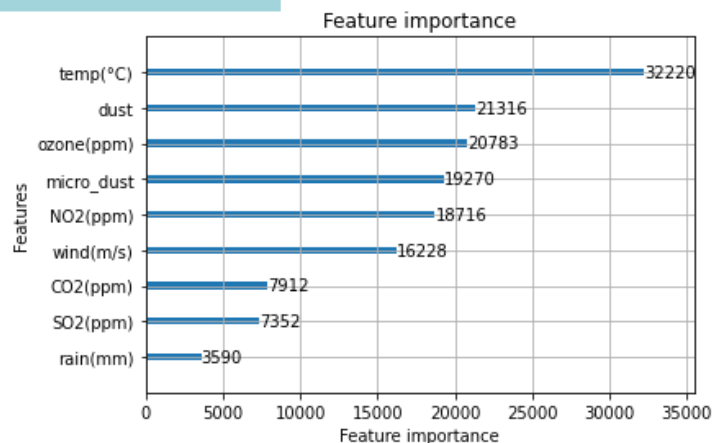
R2 score

## 회귀 모델 - LightGBM

Metric : **R2 score** (상관관계가 높을수록 1에 가까워 짐)



모든 feature



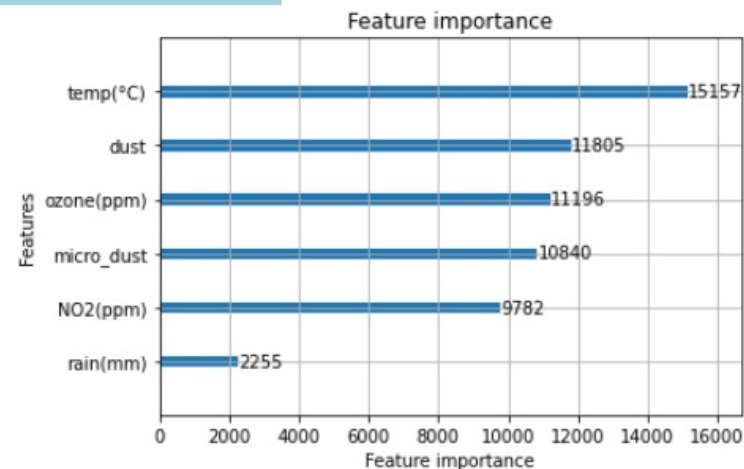
```
raw_y = df['rent_num']
raw_x = df.drop(['rent_num', 'time', 'place'], axis=1)
```

```
r2 = r2_score(test_y, predict_test)
```

```
print('R2 score: ', r2)
```

```
R2 score: 0.38498737913133607
```

Feature select



```
raw_y = df['rent_num']
raw_x = df.drop(['rent_num', 'time', 'place', 'CO2(ppm)', 'SO2(ppm)', 'wind(m/s)'], axis=1)
```

```
r2 = r2_score(test_y, predict_test)
```

```
print('R2 score: ', r2)
```

```
R2 score: 0.31615933620496073
```



구분	설명
알고리즘	LightGBM
파라미터	<code>lgb.train (Params, train_ds, 2000, test_ds, verbose_eval=100, early_stopping_rounds=200)</code>
독립변수 (X)	날씨 : 평균기온, 강수량, 평균풍속 대기오염 : 미세먼지 농도, 초미세먼지 농도, 이산화질소농도, 오존농도, 이산화탄소농도, 아황산가스농도
타겟변수 (Y)	일별 따릉이 대여횟수
R2 score	<b>0.3849</b>



## 활용방안

### 1. 따릉이의 원활한 공급

- > 공공자전거 따릉이의 이용객들의 계절, 날씨의 영향을 받음에 따라 따릉이의 수요와 공급에도 변화가 필요  
날씨와 대기의 질에 따라 따릉이의 수요를 예측하여 효율적이 따릉이 배치가 가능

### 2. 파손, 훼손 등의 대처 가능

- > 따릉이의 이용객들이 많아질수록 파손, 훼손의 위험도가 높음  
이용객들이 많은 시기가 다가오기 전에 미리 따릉이를 점검하고, 보완하여 더 많은 따릉이를 공급 할 수 있도록 대처가 가능

## 한계점



### 1. 데이터 논리성 간과

한강공원이라는 공간적 특성 고려하지 못하였다.  
특히, 휴일, 명절 등의 변수를 고려하지 않고 분석을 진행하여 데이터의 논리성 부족 문제점이 발생하였다.

또한, 2020년 따릉이 이용자 수의 폭발적인 증가로 데이터 쏠림 현상이 발생하였다.  
타 년도에 비해 많은 데이터 수로 인해 전반적인 데이터에 큰 영향을 끼치는 문제점이 발견되었다.



### 2. 데이터 확보의 한계점

'서울시 따릉이 데이터'는 다양한 유관기관에서 데이터 정보를 제공, 그에 따라 형태가 매우 다양하다.  
각 기관마다 취급하는 데이터의 종류, 형식이 모두 다르며, 기간 또한 상이하다. 그로 인해 데이터를 처리하는 데 있어 오류  
가 다수 존재하여 도출된 결과 역시 왜곡되었을 가능성이 있다.

## 참고자료

- 서울시 따릉이 분석사례(교수님 자료)



감사합니다

