2020-2학기 소프트웨어응용 프로젝트

동대문구 내 공공자전거 따릉이 대여소 방역 스케줄 추천 서비스

<10조> 2018920023 방민, 2018920031 유승리

1. 주제 선정 배경

코로나바이러스감염증-19(이하 코로나19)가 확산됨에 따라 사회적 거리두기가 만연해졌다. 하지만 출퇴근 시간에 많은 인파가 몰리는 대중교통의 경우, 사회적 거리두기가 제대로 이루어지기 쉽지 않기 때문에 자전거, 도보, 전동 킥보드 등의 방법을 선호하는 경향이 나타나고 있다. 실제로, 서울특별시 공공자전거 따릉이의 이용 건수는 전년 동기간 대비 67% 증가한 것으로 나타났다. 이에 따라, 서울시설공단 공공자전거운영처에서는 2020년 08월, '공공자전거 따릉이 방역단 운영 계획'을 발표하였으며, 매일 1,540여개 대여소를 방문하여 따릉이 이용 시 손이 자주 닿는 곳을 중점적으로 소독하는 방역단을 운영하고 있다. 그러나 사람들의 생활 패턴에 따라 특정 시간대에 따릉이 쏠림 현상이 존재하기 때문에 이를 분석하면 보다 효율적인 따릉이 방역 스케줄을 예측 및 추천할 수 있을 것이라고 생각하였으며, 서울시립대학교가 위치하고 있는 동대문구의 따릉이 대여소를 대상으로 본 서비스를 구현하고자 하였다.

2. 데이터 분석

2.1. 타겟 데이터셋

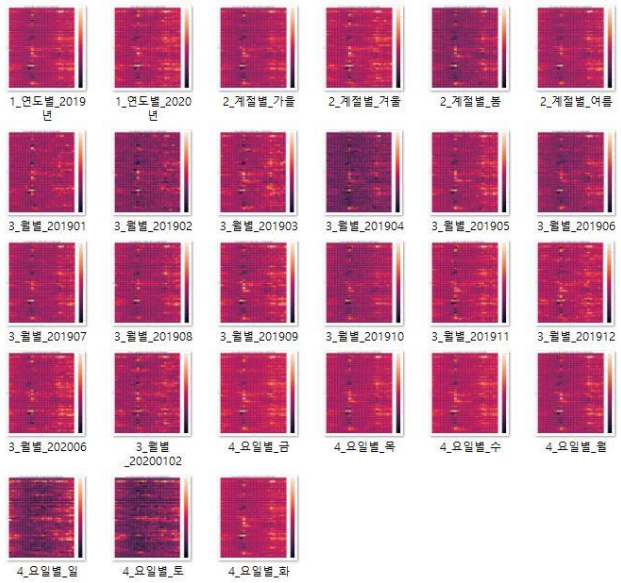
본 서비스를 구현함에 있어서 데이터 분석 단계에서 활용한 첫번째 데이터셋은 서울 열린데이터광장(http://data.seoul.go.kr/)에서 제공하는 '서울특별시 공공자전거 대여이력 정보'이다. 2019년 이후의 데이터를 모두 사용하고자 하였으나, 2020년 데이터의 경우, 01월, 02월의 절반, 06월의 데이터만 제공되고 있었기 때문에 해당 데이터만을 토대로 연구를 진행해야 했다. 이 데이터에서는 대여 일시, 대여 대여소, 반납 일시, 반납 대여소 정보를 추출하였다.

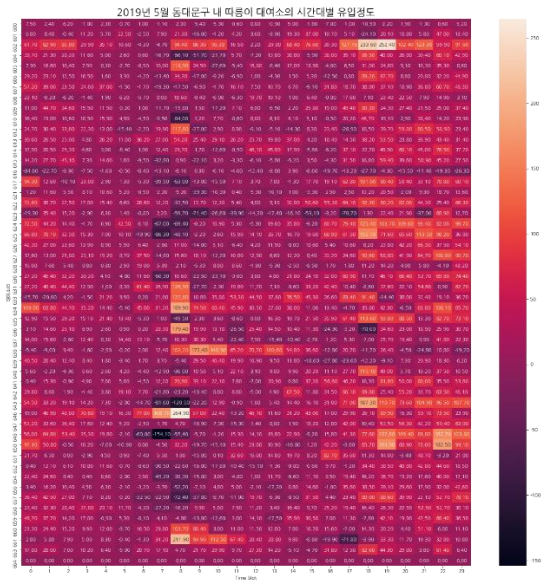
두번째 데이터셋은 서울 열린데이터광장(http://data.seoul.go.kr/)에서 제공하는 '서울특별시 공공자전거 대여소 정보'이다. 이 데이터에서는 동대문구 내 대여소의 대여소ID, 거치대수, 경도, 위도 정보를 추출하였다. 또한 이 데이터를 열람한 후, 동대문구 내 664번 대여소의 경우, 2019년 11월 말에 설치되었음을 알게 되었다. 따라서 연구에 사용하기에는 데이터의 양이 부족하다고 판단하여 첫번째 데이터셋에서 664번 대여소의 정보는 제외하였으며, 동대문구 내 나머지 51개 대여소의 정보에 대해 연구를 진행하였다.

세번째 데이터셋은 기상자료개방포털(https://data.kma.go.kr/)에서 제공하는 '서울특별시 일별 기상관측 정보'이다. 이 데이터에서는 2019-01-01 ~ 2020-06-30 기간 동안의 서울특별시의 최저 기온, 최고 기온, 일 강수량, 최대 풍속, 평균 상대습도를 추출하였다.

2.2. 분석 결과

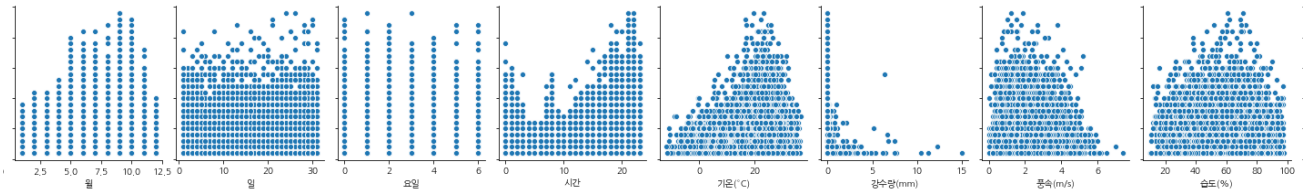
2.2.1. Heatmap



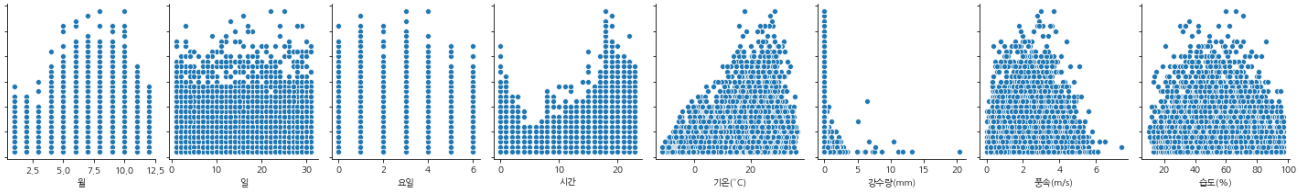


서울특별시 공공자전거 대여이력 데이터의 양상을 시각적으로 확인하기 위해서 Heatmap의 가로축을 Time Slot (0~23시), 세로축을 대여소ID로 설정하여 Heatmap을 작성하였다. Heatmap에서 나타내는 값은 대략적인 양상의 파악을 위해 유입 정도(Delta) = 0.7 * (Inflow – Outflow) + 0.3 * Inflow 식을 통해 구한 유입 정도 값으로 설정하였다. Heatmap에서 Time Slot이 밝은 색일수록 유입 정도가 크며, 어두운 색일수록 작다. 이를 통해 시간대별로 동대문구 내 각 대여소의 유입 양상이 나누어지는 것을 대략적으로 확인할 수 있었으며, 방역 시간대 추천 서비스가 효용성이 있을 것이라고 판단할 수 있었다.

2.2.2. Pair Plot



▲ 반납 건수(Inflow)에 대한 Pair Plot



▲ 대여 건수(Outflow)에 대한 Pair Plot

반납 건수(이하 Inflow) 및 대여 건수(이하 Outflow)와 관련이 있을 것으로 예상되는 파라미터인 월, 요일, 시간대, 기온(℃), 강수량(mm), 풍속(m/s), 습도(%)를 실제 Pair Plot으로 출력하였다. 해당 단계에서는 '서울특별시 일별 기상관측 정보'가 아닌 '서울특별시 시간별 기상관측 정보'를 이용하여, 각 시간대마다의 기상관측 정보를 이용하였다. 이를 통해 Inflow와 Outflow가 여러 파라미터들과 상관 관계가 있음을 대략적으로 파악할 수 있었다.

3. 머신 러닝

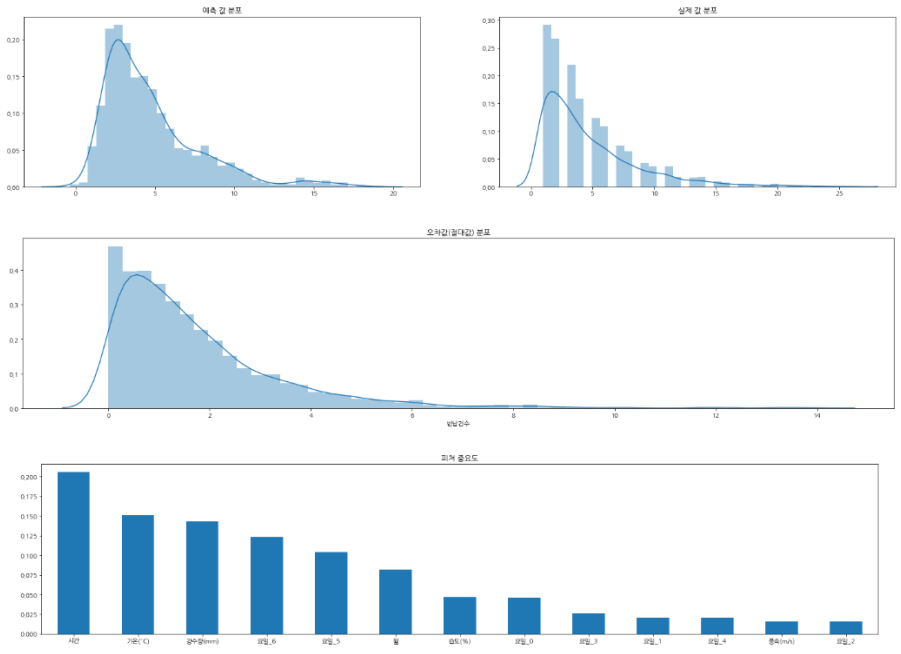
3.1. 머신 러닝 모델의 설계

3.1.1. 모델의 구조

동대문구 내 51개 대여소마다 두 개의 Regression 모델(Inflow, Outflow)을 생성한다. Inflow_[대여소ID] 모델은 해당 대여소의 시간대별 반납 건수를 예측하고, Outflow_[대여소ID] 모델은 해당 대여소의 시간대별 대여 건수를 예측한다.

3.1.2. 모델의 Input Parameter

2.2.2.에서 Inflow, Outflow 양상과 상관 관계가 있음을 확인했던 월, 요일, 시간대, 기온(℃), 강수량(mm), 풍속(m/s), 습도(%)를 머신 러닝 모델의 Input Parameter로 사용하고자 하였다. 또한, 다음 날의 일기예보는 비교적 정확한 수치이기 때문에 Input Parameter로서 사용될 수 있다고 판단하였다. 따라서 1차 학습에서는 2.2.2.에서와 동일하게 '서울특별시 시간별 기상관측 정보'를 이용하여 각 시간대마다의 기상관측 정보를 이용하였다. 이때, 대표로 이용 건수가 제일 컸던 대여소인 602번 대여소의 Inflow에 대해 학습한 결과 도출된 예측 값 분포, 실제 값 분포, 오차값 분포, 피쳐 중요도는 다음과 같다. 실제 값에 몇몇 이상치가 존재하여 오차값 또한 이상치를 포함하고 있음을 알 수 있으나, 이는 일반적인 양상보다는 테스트 데이터셋에 포함된 공휴일 등의 특수 케이스 때문이라고 판단하였다.



하지만, 각 시간대마다의 기상관측 정보를 사용하면 실제 서비스에서도 Input Parameter로 다음 날의 매 시간대마다의 일기예보 정보를 불러와야 하는데, 기상자료개방포털(https://data.kma.go.kr/)에서 제공하는 동네예보 조회 서비스 API에서는 하루 동안의 최저 기온, 최고 기온, 6시간 강수량, 풍속, 상대 습도를 제공하였다. 따라서 2차 학습에서는 머신 러닝 모델의 Input Parameter를 다음과 같이 설정하였다.

```
input_param = [ month, hour, min_temp, max_temp, rain_fall, wind_speed, humidity, day{one-hot} ]
```

month는 해당 월의 숫자로 1 ~ 12의 값이다. hour는 방역 작업이 24시간 내내 이루어지는 것이 아니라는 점을 고려하여 5 ~ 21의 값을 대입하였다. day는 요일을 one-hot 형식으로 나타낸 값으로, 월요일은 [1, 0, 0, 0, 0, 0, 0], … 일요일은 [0, 0, 0, 0, 0, 0, 1]이다. 그 외의 값은 동네예보 조회 서비스 API를 통해 불러온 다음 날의 일기 예보에서 가져온 것으로, 각각의 값에 대한 설명은 다음과 같다.

min_temp	다음 날의 최저 기온(℃)	06시에 예보됨
max_temp	다음 날의 최고 기온(℃)	15시에 예보됨
rain_fall	다음 날의 일 강수량(mm)	00시, 06시, 12시, 18시에 예보되는 6시간 예상 강수량의 총합
wind_speed	다음 날의 최대 풍속(m/s)	최댓값 구하기
humidity	다음 날의 평균 상대 습도(%)	평균값 구하기

3.2. 여러 Regression 모델의 성능 비교

학습에 사용할 Regression 모델의 종류를 결정하기 위해서 Inflow 수와 Outflow 수가 모두 제일 컸던 대여소인 602번 대여소의 데이터를 사용하여 여러 Regression 모델의 성능 비교를 수행하였다. Scikit-Learn의 linear_model 라이브러리에서 제공하는 Linear Regression 모델들과 XGBoost 모델을 학습시켜 각 모델의 MAE(Mean Absolute Error)를 출력한 결과, XGBoost 모델의 MAE 값이 약 1.73으로 가장 작았다.

```
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model as lm
import xgboost as xgb
from scipy.stats import uniform, randint

models = [
    ('lr', lm.LinearRegression(n_jobs=-1)),
    ('ridge', lm.Ridge()),
    ('lasso', lm.Lasso()),
    ('elastic', lm.ElasticNet()),
    ('LassoLars', lm.LassoLars()),
    ('LogisticRegression', lm.LogisticRegression()),
    ('SGDRegressor', lm.SGDRegressor()),
    ('Perceptron', lm.Perceptron(n_jobs=-1)),
    ('xgboost', xgb.XGBRegressor())
]
```

```
lr 2.5597978259333316
ridge 2.5597915740096853
lasso 2.565557232750446
elastic 2.562006247837826
LassoLars 2.5655572268419196
LogisticRegression 2.6759776536312847
SGDRegressor 2.989231202931475
Perceptron 3.6575418994413407
xgboost 1.7322861785828734
```

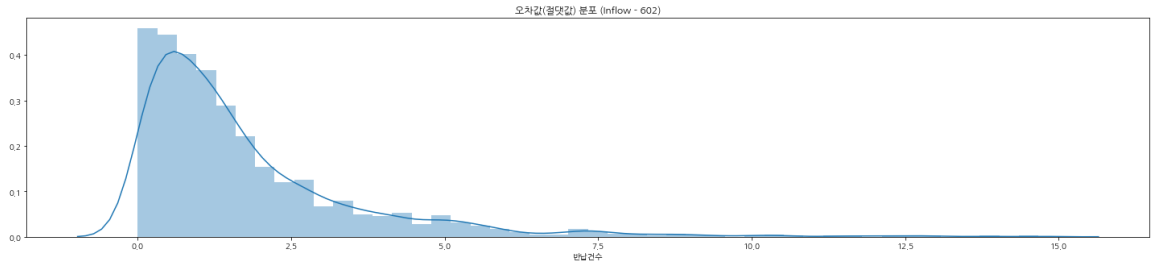
3.3. XGBoost Regression 학습

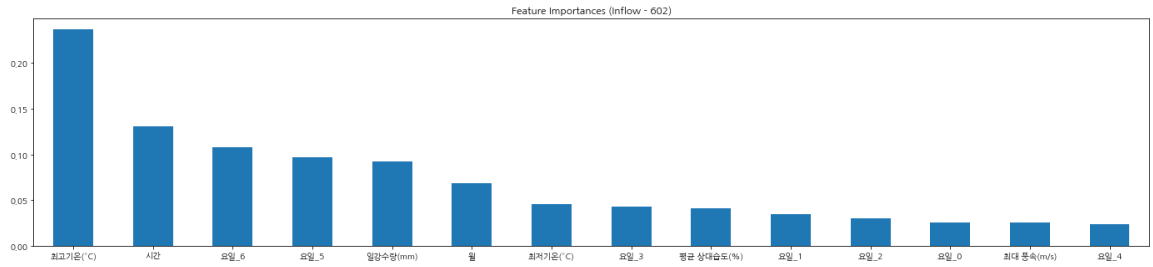
3.2.의 결과에 따라서 XGBoost Regression 학습을 통해 동대문구 내 51개 대여소 각각에 대한 Inflow와 Outflow 예측 모델을 생성 후 저장하였다. 저장한 모델의 이름은 Inflow_[대여소ID], Outflow[대여소ID]이다.

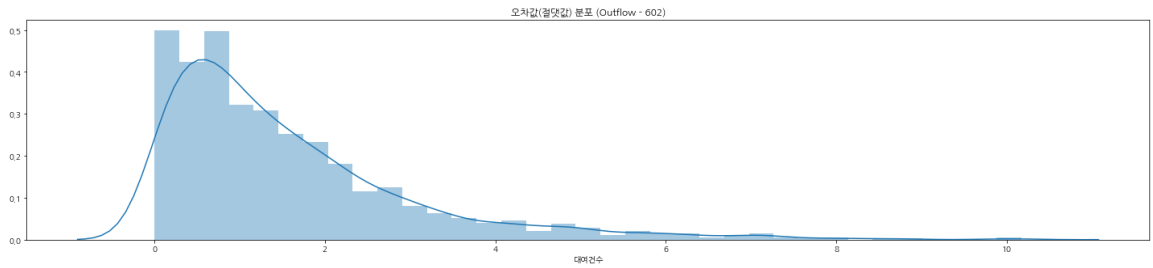
```
cols = ['월', '요일', '시간', '최저기온(°C)', '최고기온(°C)', '일강수량(mm)', '최대 풍속(m/s)', '평균 상대습도(%)']

Result_Inflow = {}
total = Bike_Inflow['반납대여소ID'].nunique()

for idx, (name, df) in enumerate(Bike_Inflow.groupby('반납대여소ID'), 1):
    print("[%d/%d] %s" %(idx, total, name), end=' ')

    try:
        df = df.groupby(cols)['반납건수'].sum().reset_index()
        df = df.join(pd.get_dummies(df['요일'], prefix="요일"))

        print(df.shape)

        x_train, x_test, y_train, y_test = train_test_split(df[features], df['반납건수'], test_size=0.2, random_state=42)

        param_grid = {
            "gamma": uniform(0, 0.5).rvs(n),
            "max_depth": range(2, 7), # default 3
            "n_estimators": randint(100, 150).rvs(n), # default 100
        }

        grid = GridSearchCV(xgb.XGBRegressor(objective ='reg:squarederror'), cv=5, n_jobs=-1, param_grid=param_grid)
        grid = grid.fit(x_train, y_train)

        model = grid.best_estimator_
        predictions = model.predict(x_test)
        mae = mean_absolute_error(y_test, predictions)

        Result_Inflow[str(name)] = {}
        Result_Inflow[str(name)]['model'] = model
        Result_Inflow[str(name)]['mae'] = mae
        Result_Inflow[str(name)]['errors'] = predictions - y_test

    except:
        continue
```

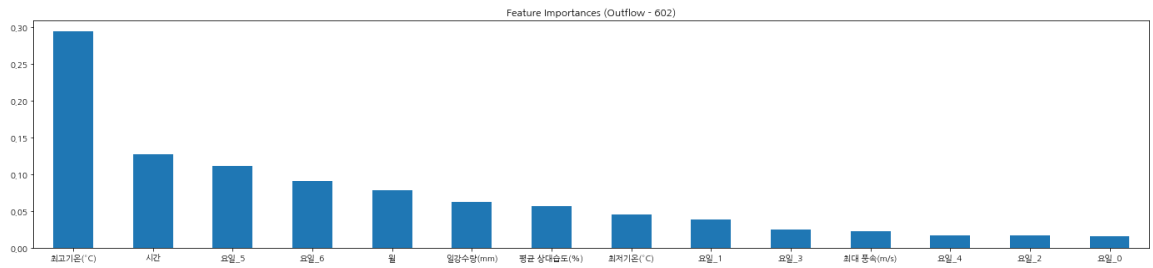
3.3.1. XGBoost Regression 학습 결과

Inflow 모델 51개의 평균 MAE는 약 1.01로 계산되었으며, Outflow 모델 51개의 평균 MAE는 약 1.08로 계산되었다. 따라서 해당 모델이 Inflow 및 Outflow를 어느 정도 잘 예측한다고 판단하였다. 또한, Inflow 수와 Outflow 수가 모두 제일 컸던 대여소인 602번 대여소의 모델인 Inflow_602, Outflow_602를 사용하여 각 모델의 오차값 분포와 피쳐 중요도를 출력했다. 마찬가지로 오차값 분포에서 몇 이상치가 발견되었으나 공휴일 등의 예외 상황이라고 판단하였으며, 피쳐 중요도에서는 대여소의 단위 시간당 Inflow와 Outflow에 영향을 미치는 요소 TOP 5는 최고 기온(℃), 시간대, 요일, 일 강수량(mm), 월이었음을 알 수 있었다.









3.3.2. XGBoost Regression 예측 결과

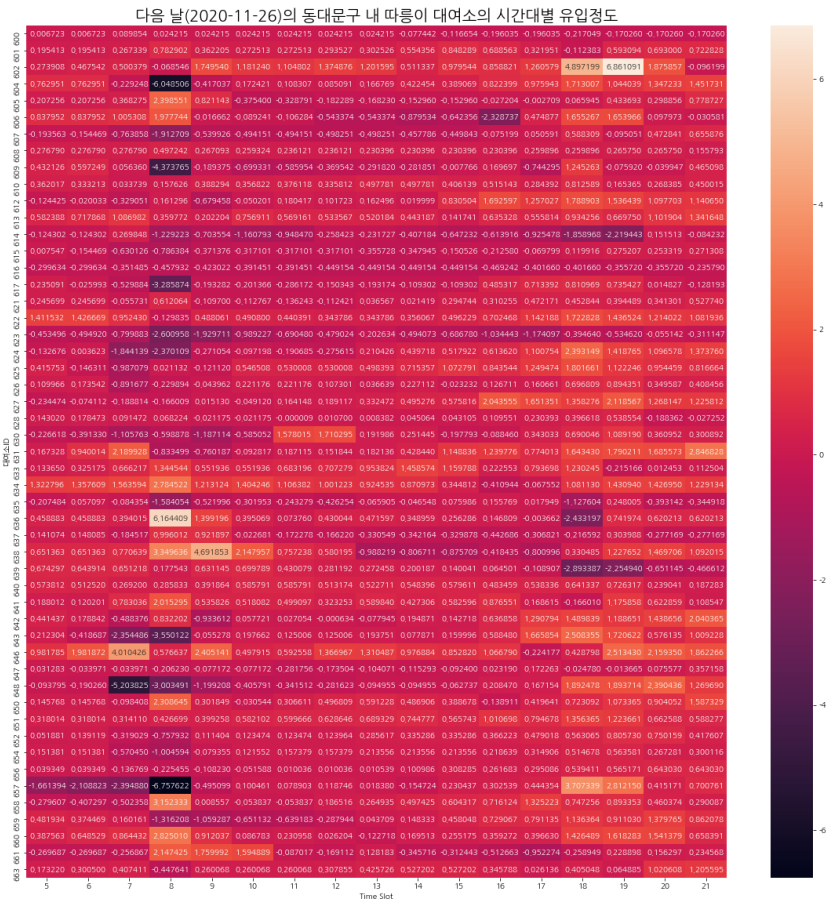
학습된 Inflow, Outflow 모델을 토대로 예측된 각 대여소의 시간대별 Inflow, Outflow 값을 통해 유입 정도(이하 Delta)를 계산한다. Delta를 계산하는 식은 다음과 같다.

$$\text{유입 정도(Delta)} = \alpha \times (\text{Inflow} - \text{Outflow}) + (1 - \alpha) \times \text{Inflow}, \quad \text{이때, } \alpha = 0.85$$

Delta를 계산하는 식을 위와 같이 설계한 이유는 대여소의 관점에서 각 시간대마다 실제로 유입되는 대수인 Inflow에

서 유출되는 대수인 Outflow를 뺀 값이 실제로 사람이 사용하고 반납한 따릉이 대수, 즉 소독이 필요한 대수이기 때문이다. 또한, 유입되는 대수 Inflow 값의 크기를 추가로 반영하여 절대적인 Inflow 값도 고려하도록 하였다. 따라서 각 시간대별로 Delta가 큰 대여소를 방문 후 소독하는 것이 효율적이게 된다. α 값으로 가장 적절한 수에 대해서는 많은 실험을 진행하지는 못하였으나 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9의 값 중에서 실제로 Heatmap을 출력했을 때 제일 양상이 두드러지게 분리되어 나타나는 값이었던 0.85를 선택하였다. 하지만 아직 Delta를 구하는 식에는 허점이 많기 때문에, 추후 연구에서 여러 α 값에 대해 방역 스케줄을 추천한 후 예상 유효 소독 대수를 계산하여 가장 소독 대수가 큰 α 값을 찾아보는 등의 방법으로 Delta를 계산하는 식을 개선할 수 있을 것으로 보인다.

이와 같이 계산한 Delta 값을 토대로 다음 날의 유입 정도에 따른 Heatmap을 출력할 수 있다.



4. 스케줄 추천 알고리즘 설계

4.1. 필요한 정보

4.1.1. 각 대여소 간 자동차로 걸리는 시간

tmap(http://tmapapi.sktelecom.com/)에서 제공하는 자동차 경로안내 API를 이용하여 각 대여소 간의 경로, 즉 51×51개의 경로에 대한 소요 시간을 구하였다. 그 결과, 최소 소요 시간이 35초(608번 대여소 → 616번 대여소)인 반면 최대 소요 시간이 2814초(613번 대여소 → 607번 대여소)로 그 차이가 매우 컸다. 하지만 51×51개의 소요 시간 정보, 각 대여소별 Delta 값이 큰 시간대, Delta 값의 크기를 모두 고려하여 방역 스케줄을 도출하는 데에 어려움을 겪었기 때문에 51×51개의 소요 시간의 평균 값으로 모든 경로의 소요 시간을 고정하기로 하였다. 따라서 소요 시간의 평균 값인 836.969초(= 13분 56.969초)를 어림잡아 약 15분으로 가정하기로 하였다.

4.1.2. 예측 결과를 활용한 정보

top_13_sorted	Delta 값의 합이 큰 TOP 13 대여소의 ID
desired_order	각 대여소마다 Delta 값이 큰 순서로 (time slot, 유입 정도) 정렬
largest_stop	각 시간대마다 그 시간대가 제일 유입 정도가 큰 대여소의 ID
visit_order	각 대여소를 방문한 횟수

4.2. 스케줄 추천 알고리즘

4.2.1. 가정

방역에 참여하는 팀은 2개(A팀, B팀)이다. 대여소간 소요 시간과 대여소별 소독 시간은 모두 15분으로 가정한다. 따라서 한 팀은 한 시간마다 2개의 대여소를 소독이 가능하므로, 두 팀은 총 4개의 대여소를 소독할 수 있다. 그리고 소독은 이전 Time Slot의 Delta 값을 토대로 수행한다. 예를 들어, Time Slot 8(08~09시)에 Delta의 크기가 가장 큰 대여소는 Time Slot 9(09~10시)에 소독한다. 즉, 유입 정도의 예측은 05~21시에 대해서, 방역 스케줄 추천은 06~22시에 대해서 이루어진다. 또한, TOP 13 대여소는 한 번 더 소독한다.

4.2.2. 각 Time Slot에 소독할 대여소 배정

각 Time Slot에 소독할 대여소를 배정하는 순서는 다음과 같으며, 이는 모두 **visit_order**에 저장된다.

(1) 직전 Time Slot에서 제일 Delta 값이 컸던 대여소들을 배정한다. (최대 4개)

(2) 제일 Delta 값이 컸던 대여소의 개수가 4개가 넘는 Time Slot의 경우, 그 다음 Time Slot으로 가득 채워 배정한다.

(3) Time Slot 21의 배정이 끝난 경우, 방문 횟수(**visit_count**)가 0인 대여소와 TOP 13 대여소를 무작위로 남는 Time Slot에 배정한다.

(4) A팀에는 각 Time Slot의 홀수 번째 대여소, B팀에는 각 Time Slot의 짝수 번째 대여소를 배정한다.

```
In [272]: visit_order

Out[272]: {'6': ['616', '612', '622', '604'],
           '7': ['642', '651', '609', '610'],
           '8': ['646', '631', '614', '600'],
           '9': ['636', '658', '660', '634'],
           '10': ['638', '605', '650', '661'],
           '11': ['639', '641', '606', '637'],
           '12': ['621', '608', '617', '640'],
           '13': ['630', '656', '607', '654'],
           '14': ['628', '615', '635', '623'],
           '15': ['633', '602', '646', '634'],
           '16': ['601', '622', '638', '631'],
           '17': ['660', '651', '613', '627'],
           '18': ['647', '633', '636', '625'],
           '19': ['657', '643', '624', '625'],
           '20': ['602', '627', '626', '652'],
           '21': ['648', '659', '613', '663']}
```

```
In [274]: # team_A와 team_B에게 배분
team_A = {}
team_B = {}

for t in range(6, 22):
    team_A[str(t)] = []
    team_B[str(t)] = []
    team_A[str(t)].append(visit_order[str(t)][0])
    team_A[str(t)].append(visit_order[str(t)][2])
    team_B[str(t)].append(visit_order[str(t)][1])
    team_B[str(t)].append(visit_order[str(t)][3])

print(team_A)
print(team_B)

{'6': ['616', '622'], '7': ['642', '609'], '8': ['646', '614'],
'9': ['636', '660'], '10': ['638', '650'], '11': ['639', '606'],
'12': ['621', '617'], '13': ['630', '607'], '14': ['628', '635'],
'15': ['633', '646'], '16': ['601', '638'], '17': ['660', '613'],
'18': ['647', '636'], '19': ['657', '624'], '20': ['602', '626'],
'21': ['648', '613']}
{'6': ['612', '604'], '7': ['651', '610'], '8': ['631', '600'],
'9': ['658', '634'], '10': ['605', '661'], '11': ['641', '637'],
'12': ['608', '640'], '13': ['656', '654'], '14': ['615', '623'],
'15': ['602', '634'], '16': ['622', '631'], '17': ['651', '627'],
'18': ['633', '625'], '19': ['643', '625'], '20': ['627', '652'],
'21': ['659', '663']}
```

4.3. 효용성 검증

본 스케줄 추천 알고리즘의 효용성을 검증하기 위해 추천 스케줄과 랜덤 스케줄을 비교한다.

4.3.1. 예상 유효 소독 대수

추천된 방역 스케줄과 랜덤으로 짜여진 방역 스케줄의 효용성을 비교하는 척도로 예상 유효 소독 대수를 사용한다. 예상 유효 소독 대수를 계산하는 식은 다음과 같다.

$$\text{예상 유효 소독 대수} = \sum \{(\text{각 대여소의 기존 거치대수}) + (\text{각 대여소가 소독되기 직전 Time Slot까지의 (Inflow} - \text{Outflow) 값의 합})\} = \sum (\texttt{disinfected[stop\_id]})$$

이때, `disinfected[stop_id]` < 0인 경우, 0으로 처리한다.

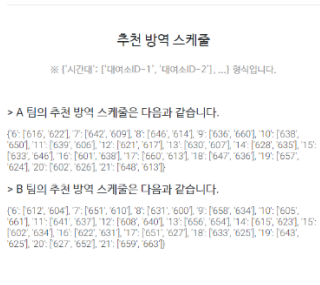
4.3.2. 랜덤 스케줄과의 비교

추천 방역 스케줄의 예상 유효 소독 대수가 다음 경우에서 모두 랜덤 방역 스케줄의 예상 유효 소독 대수보다 컸다.

(1) 2020-11-26-목

`input_param = [ 11, {hour}, 3.0, 10.0, 0.0, 1.7, 67.5, 3{one-hot} ]`

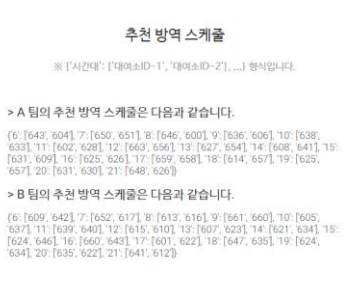
추천 방역 스케줄	랜덤 방역 스케줄


629.29대
	In [80]:
	`disinfected_day_ran = 0`
	`for stop_id in Bike_Stop_ID:`
	`  #print(disinfected_ran[str(stop_id)])`
	`  if disinfected_ran[str(stop_id)] > 0:`
	`  disinfected_day_ran += disinfected_ran[str(stop_id)]`
	`print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)`
	랜덤 경로의 하루 예상 유효 소독 대수: 588.2883203625679
	588.29대
	
	In [88]:
	`disinfected_day_ran = 0`
	`for stop_id in Bike_Stop_ID:`
	`  #print(disinfected_ran[str(stop_id)])`
	`  if disinfected_ran[str(stop_id)] > 0:`
	`  disinfected_day_ran += disinfected_ran[str(stop_id)]`
	`print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)`
	랜덤 경로의 하루 예상 유효 소독 대수: 581.8257563710213
	581.83대

(2) 2020-08-14-금

`input_param = [8, {hour}, 26.5, 28.0, 0.4, 3.5, 86.1, 4{one-hot} ]`

추천 방역 스케줄	랜덤 방역 스케줄


619.4대
	In [119]:
	`disinfected_day_ran = 0`
	`for stop_id in Bike_Stop_ID:`
	`  #print(disinfected_ran[str(stop_id)])`
	`  if disinfected_ran[str(stop_id)] > 0:`
	`  disinfected_day_ran += disinfected_ran[str(stop_id)]`
	`print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)`
	랜덤 경로의 하루 예상 유효 소독 대수: 581.2920391559601
	581.29대
	
	In [127]:
	`disinfected_day_ran = 0`
	`for stop_id in Bike_Stop_ID:`
	`  #print(disinfected_ran[str(stop_id)])`
	`  if disinfected_ran[str(stop_id)] > 0:`
	`  disinfected_day_ran += disinfected_ran[str(stop_id)]`
	`print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)`
	랜덤 경로의 하루 예상 유효 소독 대수: 569.8832812309265
	569.88대

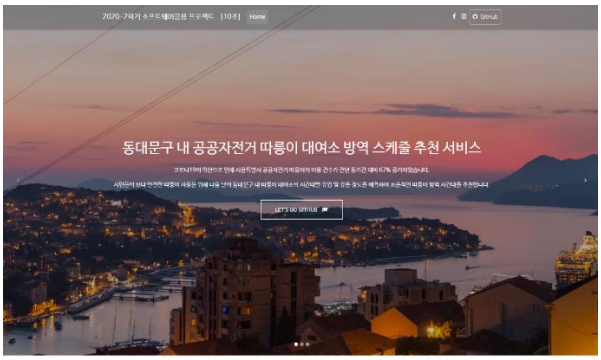
(3) 2020-10-11-일

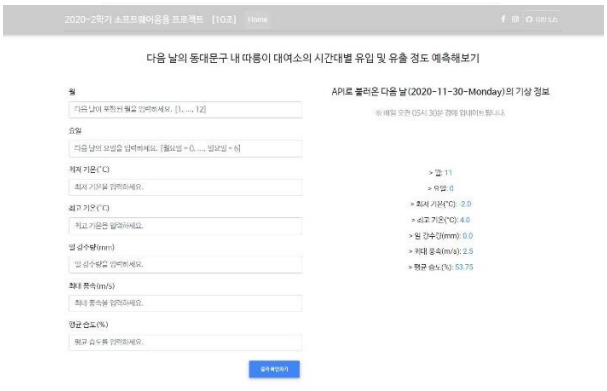
input_param = [10, {hour}, 12.7, 20.8, 0.0, 1.8, 68.5, 6{one-hot}]	
추천 방역 스케줄	랜덤 방역 스케줄

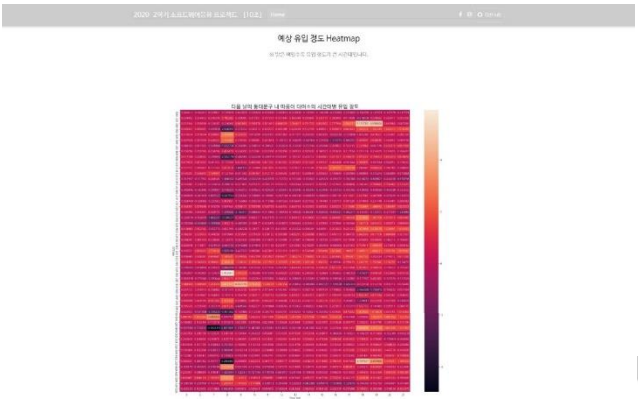
추천 방역 스케줄 예상 유효 소독 대수
※ {'시간대': ['대여소ID-1', '대여소ID-2'], ...} 형식입니다.
> A 팀의 추천 방역 스케줄은 다음과 같습니다.
{6: ['624', '654'], 7: ['622', '651'], 8: ['646', '600'], 9: ['636', '606'], 10: ['638', '658'], 11: ['639', '608'], 12: ['631', '657'], 13: ['630', '614'], 14: ['628', '615'], 15: ['663', '634'], 16: ['602', '641'], 17: ['660', '651'], 18: ['643', '640'], 19: ['612', '609'], 20: ['634', '627'], 21: ['648', '660']}
> B 팀의 추천 방역 스케줄은 다음과 같습니다.
{6: ['623', '616'], 7: ['642', '656'], 8: ['613', '652'], 9: ['661', '641'], 10: ['637', '625'], 11: ['605', '626'], 12: ['601', '607'], 13: ['610', '647'], 14: ['621', '602'], 15: ['622', '648'], 16: ['646', '630'], 17: ['661', '627'], 18: ['650', '613'], 19: ['604', '633'], 20: ['617', '635'], 21: ['663', '659']}
600.23대
	In [135]:
	disinfected_day_ran = 0
	for stop_id in Bike_Stop_ID:
	#print(disinfected_ran[str(stop_id)])
	if disinfected_ran[str(stop_id)] > 0:
	disinfected_day_ran += disinfected_ran[str(stop_id)]
	print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)
	랜덤 경로의 하루 예상 유효 소독 대수: 565.515043258667
	565.52대
	In [143]:
	disinfected_day_ran = 0
	for stop_id in Bike_Stop_ID:
	#print(disinfected_ran[str(stop_id)])
	if disinfected_ran[str(stop_id)] > 0:
	disinfected_day_ran += disinfected_ran[str(stop_id)]
	print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)
	랜덤 경로의 하루 예상 유효 소독 대수: 577.5639467835426
	577.56대

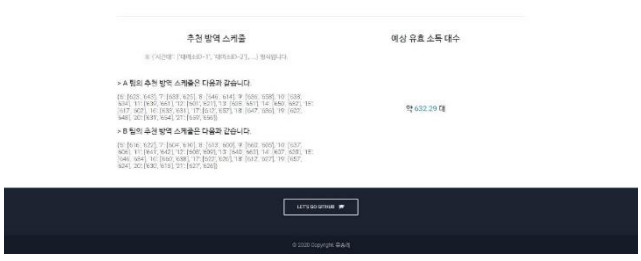
5. 웹 서비스 구축

웹 서비스는 파이썬으로 작성된 웹 프레임워크 중 하나인 Flask를 사용하여 제작하였고, server.py 파일 내에 머신 러닝 학습 이후의 모든 동작을 구현하였다. 템플릿인 index.html 파일은 MDBootstrap (https://mdbootstrap.com/freebies/)에서 제공하는 무료 템플릿을 기반으로 제작하였다.









6. 고찰 및 확장 가능성

- 파라미터를 추가하여 더 많은 횟수의 실험을 통해 최적의 유입 정도를 계산하는 식을 도출해낸다면 추천 방역 스케줄의 정확도를 향상시킬 수 있을 것이다.
- 51개의 대여소 간 소요 시간의 평균값이 아닌 51×51개의 소요 시간, 각 대여소별 유입 정도가 큰 시간대, 유입 정도의 크기를 모두 고려하여 방역 스케줄을 도출해내도록 알고리즘을 개선한다면 대략적인 시간(hour) 단위가 아닌 minute 단위의 스케줄 추천이 가능할 것이다.
- 코로나19 이후 따릉이 이용 건수는 전년 동기간 대비 67% 증가했으나, 공공데이터포털에는 2020년 1월, 2월 절반, 6월의 따릉이 대여이력 데이터만 업로드 되어 있다. 따라서 증가된 이용 건수를 학습 모델에 반영하기에는 데이터가 부족했을 것으로 예상된다.
- 본 서비스에서 머신 러닝 모델을 만들 때, 여러 종류의 Regression 모델 중에서 실험 결과 가장 성능이 좋았던 XGBoost를 선택했으나 여러 이상치까지 예측을 할 수 있도록 LSTM 등의 딥러닝 모델로도 학습을 진행하여 성능 비교를 해볼 수 있을 것이다.
- 보다 정확한 다음 날의 일기 예보를 토대로 각 대여소의 시간대별 유입 정도와 유출 정도를 예측할 수 있으므로 따릉이 재배치 솔루션 혹은 따릉이 관련 데이터 연구에 적용될 수 있을 것으로 보인다. 또한, 동대문구와 따릉이 이용량이 많은 마포구, 영등포구, 송파구뿐만 아니라 서울시 내 모든 구에 대해서도 확장이 가능할 것이다.

7. 참고 자료

- 서울시설공단 공공자전거 운영처, 『코로나19 극복 희망일자리사업』 공공자전거 따릉이 방역단 운영 계획
- 기상청, 동네예보 조회서비스 Open API 활용가이드
- 서울 열린데이터 광장 - http://data.seoul.go.kr/
- 기상청 - https://www.weather.go.kr/
- 기상자료개방포털 - https://data.kma.go.kr/
- tmap API - http://tmapapi.sktelecom.com/
- MDBootstrap - https://mdbootstrap.com/freebies/
- All about 따릉이 EDA - https://dailyheumsi.tistory.com/
- 다변인 선형회귀를 활용한 배추 가격 예측 AI 개발하기

  - https://ndb796.tistory.com/ , https://github.com/ndb796/Vegita/