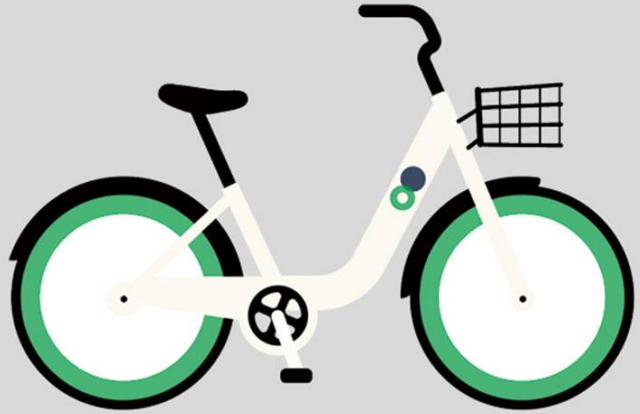


소프트웨어응용 프로젝트 최종 발표

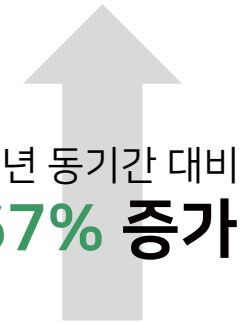
동대문구 내 공공자전거 따릉이 대여소 방역 스케줄 추천 서비스

<10조> 2018920023 방민 2018920031 유승리

목차

주제 선정 배경

코로나19 확산으로 인한 사회적 거리두기



전년 동기간 대비
67% 증가

서울시 공공자전거 따릉이 이용 건수 증가

<서울시 공공자전거 따릉이 방역 강화>

- 매일 1,540여개 대여소 방문 후 전수 소독
- 하지만 특정 시간대에 따릉이 쏠림 현상 존재

- 소독대상 : 따릉이 이용시 손이 자주 닿는 곳 중점 소독

  자전거 … 손잡이, 단말기, 안장 / 거치대 … 잠금장치

핸들	단말기(QR)	단말기(LCD)	안장	잠금장치

* [공공자전거 따릉이 방역단 운영 계획]에서 발췌

“보다 효율적인 서울시 공공자전거 따릉이 방역 스케줄을 예측 및 추천”

타겟 데이터셋

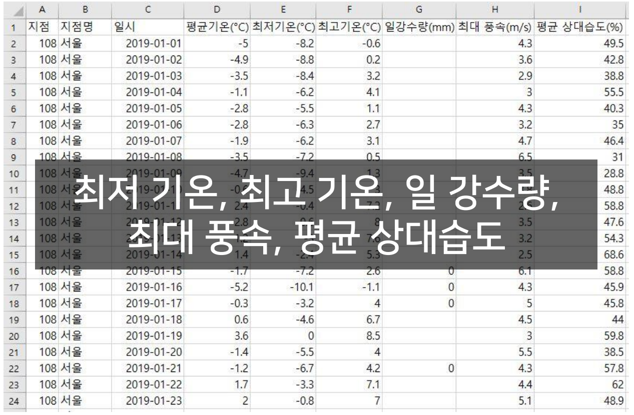
서울특별시 공공자전거 대여이력 정보
(2019-01-01 ~ 2020-06-30)

* 2020년 데이터의 경우, 01월, 02월 절반, 06월의 데이터만 제공됨

* 664번 대여소의 경우, 2019년 11월 말에 설치되어 데이터 부족으로 제외함

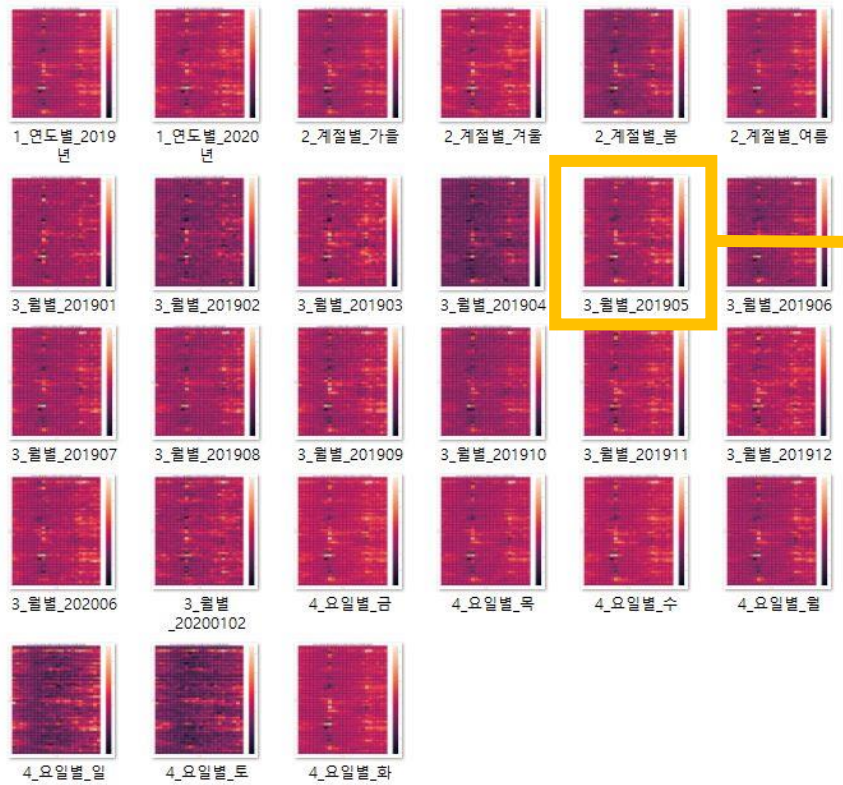
서울특별시 공공자전거 대여소 정보

서울특별시 일별 기상관측 정보
(2019-01-01 ~ 2020-06-30)

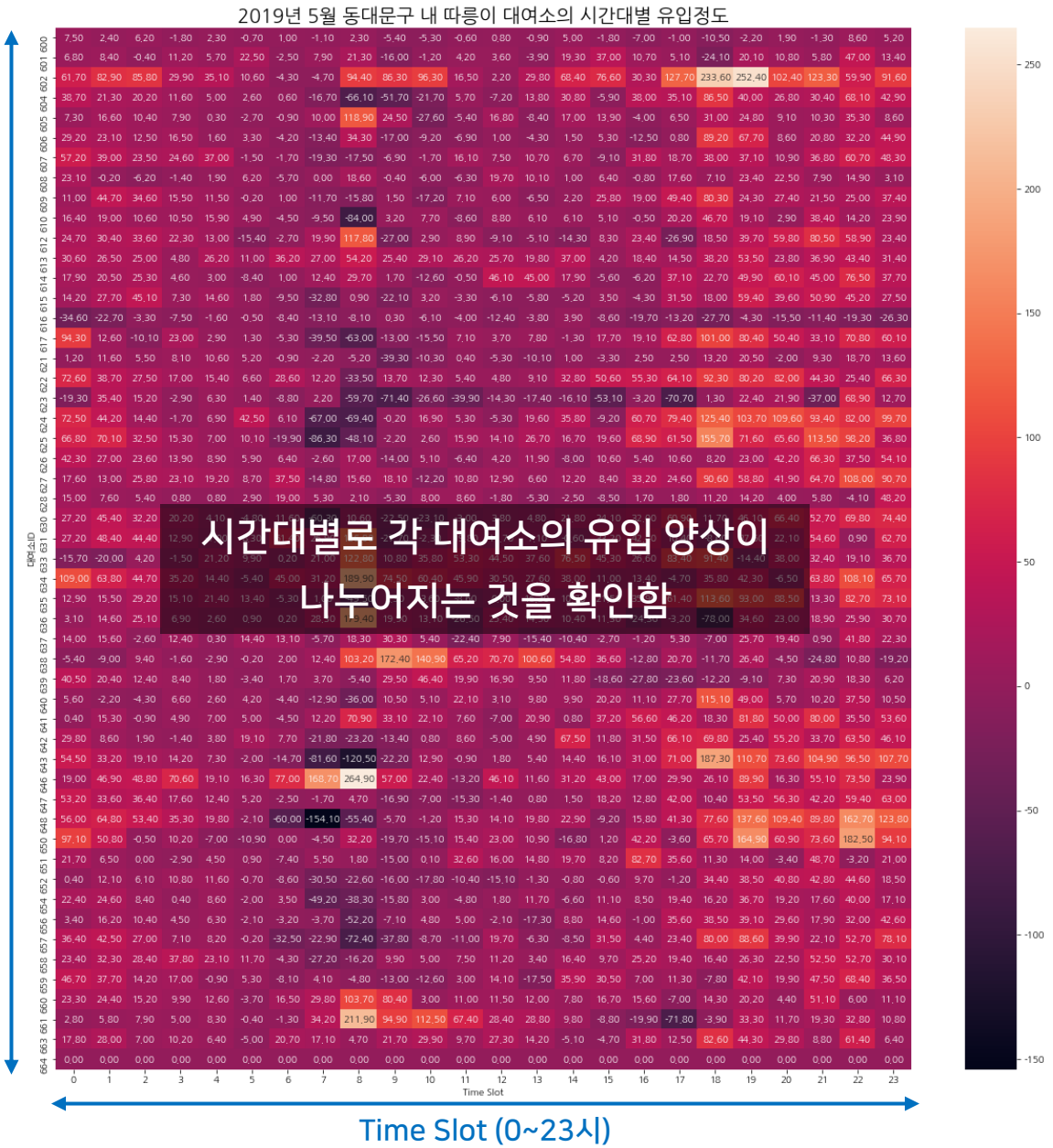


* 서울 열린데이터 광장(http://data.seoul.go.kr/), 기상자료개방포털(https://data.kma.go.kr/) 제공

분석 결과 – Heatmap

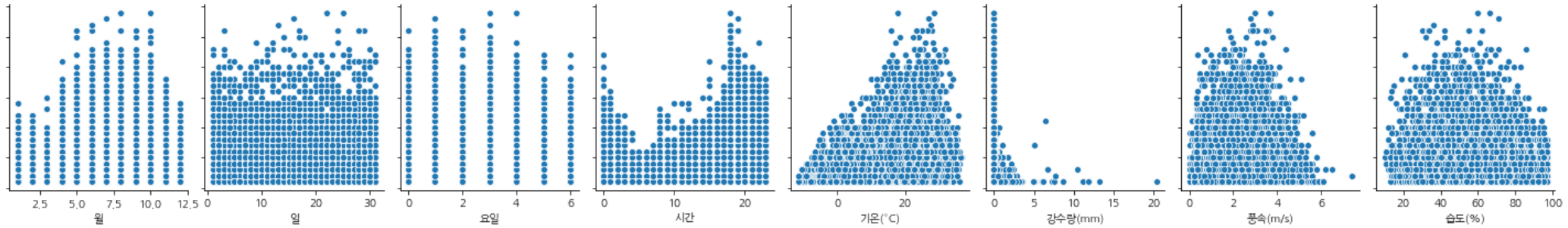


- Time Slot이 밝은 색일수록 유입 ↑
- Time Slot이 어두운 색일수록 유입 ↓

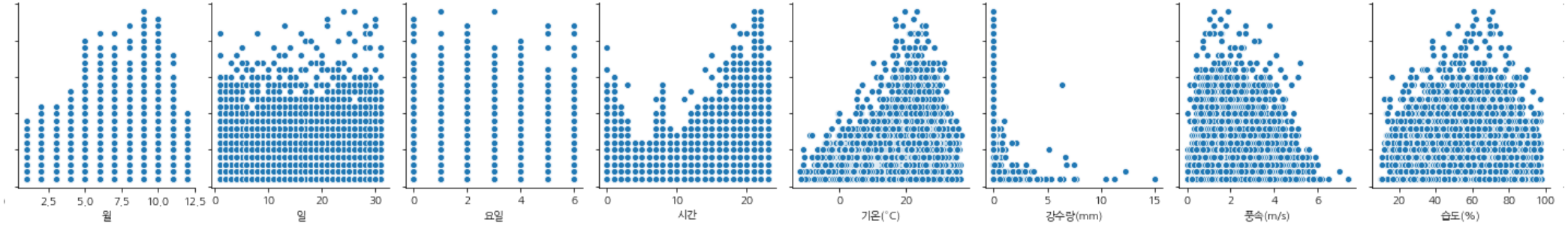


분석 결과 – Pair Plot

반납 건수
(Inflow)



대여 건수
(Outflow)



Inflow & Outflow가 여러 파라미터들과 상관 관계가 있음을 대략적으로 파악함

모델의 Input Parameter

월 = [1, 0, 0, 0, 0, 0, 0]
화 = [0, 1, 0, 0, 0, 0, 0]
:
일 = [0, 0, 0, 0, 0, 0, 1]

1 ~ 12(월) 5 ~ 21(시)

동네예보 조회 서비스 API (다음 날의 일기 예보)

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

다음 날의 최저 기온(℃)
(06:00에 예보됨)

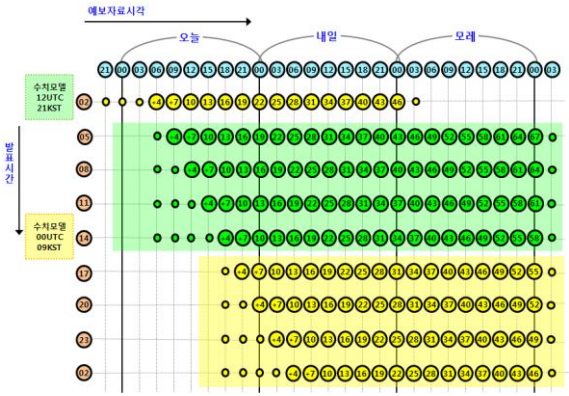
다음 날의 최고 기온(℃)
(15:00에 예보됨)

다음 날의 일 강수량(mm)
(00:00, 06:00, 12:00, 18:00에
예보되는 6시간 예상 강수량의 총합)

다음 날의 최대 풍속(m/s)
(최댓값 구하기)

다음 날의 평균 상대 습도(%)
(평균값 구하기)

[동네예보 현업운영 발표시간 별 예보자료시각]



* 동네예보 조회 서비스 오픈 API: 기상자료개방포털(https://data.kma.go.kr/) 제공

여러 Regression 모델의 성능 비교

Scikit-Learn의 **linear_model 라이브러리**에 포함된 Linear Regression 모델과 **XGBoost** 모델의 학습

```
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())
]
```

```
best_model, best_mae = None, float('inf')
for model_name, model in models:
    param_grid = params[model_name]
    grid = GridSearchCV(model, 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)

    print(model_name, mae)

    if mae < best_mae:
        best_model = model
```

각 Regression 모델의 **MAE(Mean Absolute Error)** 비교

```
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
```

XGBoost 모델의 MAE 값이 가장 작았음

```
In [56]: best_model
Out[56]: XGBRegressor(gamma=0.24714164807668376, max_depth=5, n_estimators=119)
```

* 이용량이 제일 많았던 ID=602 대여소의 Inflow 데이터로 실험함

XGBoost Regression 학습

```
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
```

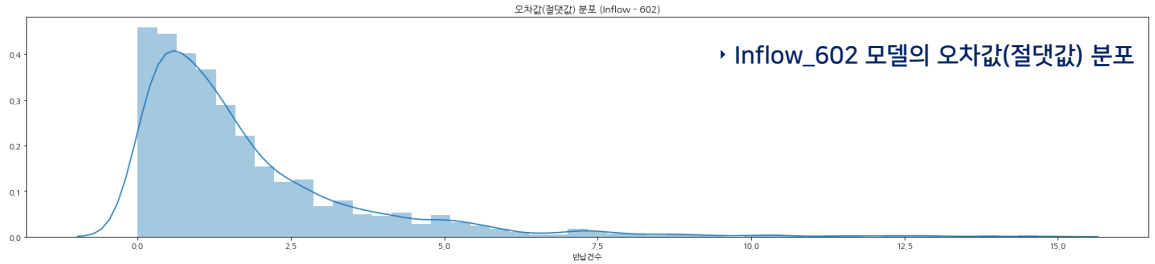
XGBoost Regression 학습 결과

{ Inflow 모델 }

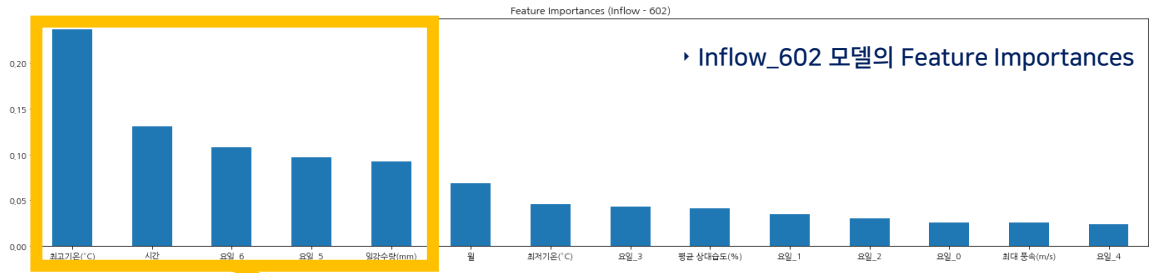
In [53]:
```
errors = np.array([result['mae'] for result in Result_Inflow.values()])
errors.mean()
```

Out[53]: `1.0110311263420817`

‣ Inflow 모델 51개의 평균 MAE(Mean Absolute Error)



‣ Inflow_602 모델의 오차값(절댓값) 분포



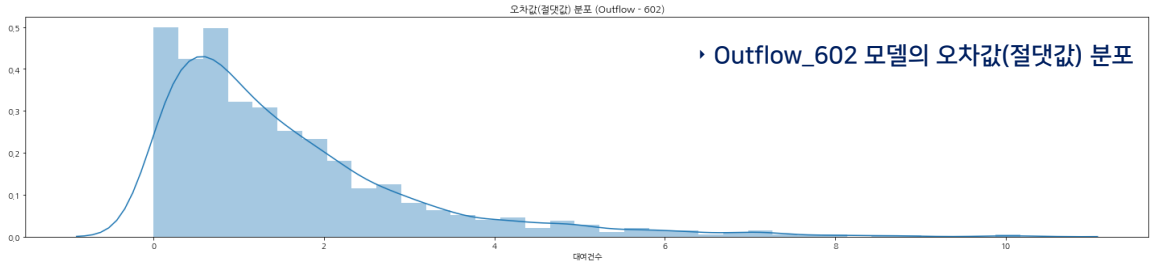
‣ Inflow_602 모델의 Feature Importances

{ Outflow 모델 }

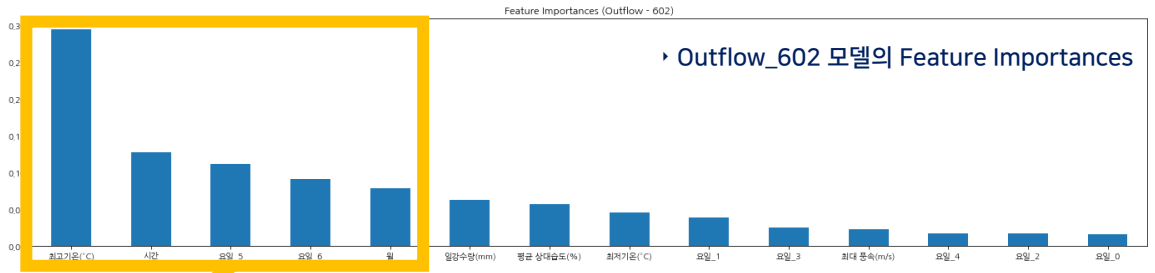
In [58]:
```
errors = np.array([result['mae'] for result in Result_Outflow.values()])
errors.mean()
```

Out[58]: `1.0802892975824931`

‣ Outflow 모델 51개의 평균 MAE(Mean Absolute Error)



‣ Outflow_602 모델의 오차값(절댓값) 분포



‣ Outflow_602 모델의 Feature Importances

* 대여소의 단위 시간당 Inflow와 Outflow에 영향을 미치는 요소 TOP 5

: 최고 기온, 시간대, 요일, 일 강수량, 월

XGBoost Regression 예측 결과

예측된 Inflow & Outflow 값을 통해 계산한 다음 날의 **유입 정도(Delta)**를 Data Frame으로 저장

In [47]:
```
Result2 = pd.DataFrame(Result_All2['600'])
for id_stop in Bike_Stop_ID:
    if id_stop == 600:
        continue
    tmppp = pd.DataFrame(Result_All2[str(id_stop)])
    Result2 = pd.concat([Result2, tmppp])

Result2 = Result2.reset_index(drop = True)
Result2
```

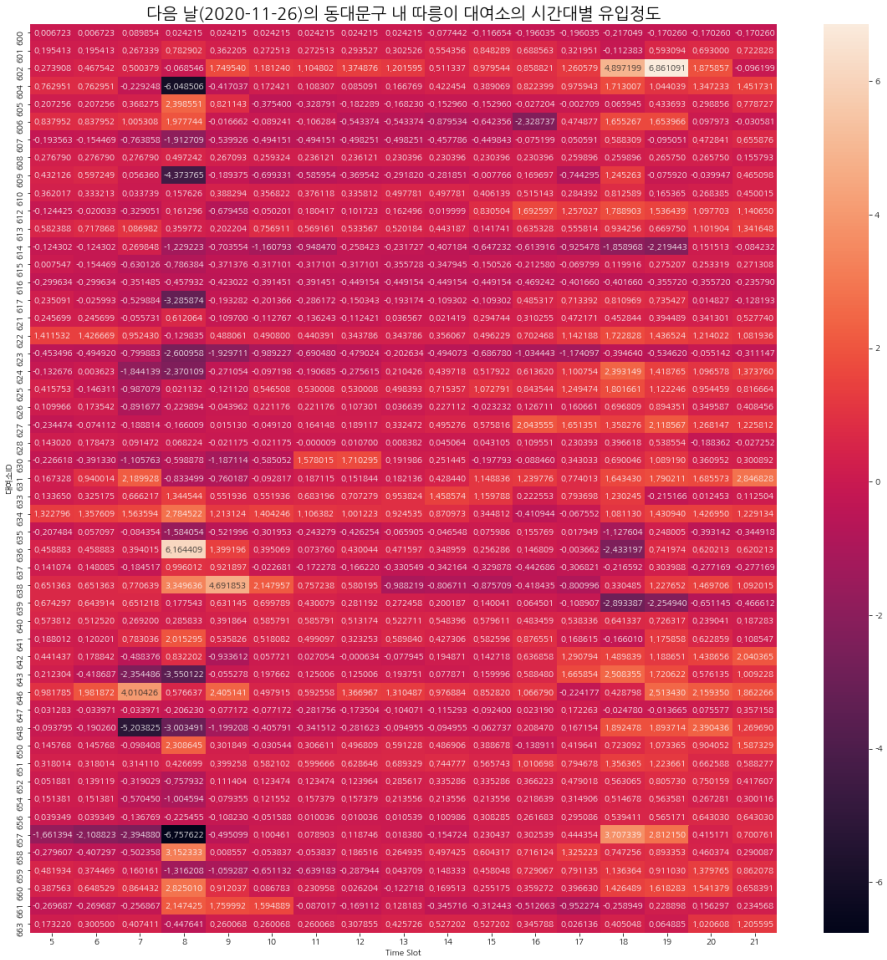
Out[47]:

	대여소ID	Time Slot	Inflow	Outflow	Delta
0	600	5	1.143319	1.337173	0.006723
1	600	6	1.143319	1.337173	0.006723
2	600	7	1.226451	1.337173	0.089854
3	600	8	1.226451	1.414396	0.024215
4	600	9	1.226451	1.414396	0.024215
...	...	...	...	...	...
862	663	17	2.535761	2.952500	0.026136
863	663	18	3.017593	3.073582	0.405048
864	663	19	2.642195	3.032130	0.064885
865	663	20	3.175183	2.534794	1.020608
866	663	21	3.416070	2.600559	1.205595

867 rows × 5 columns

* 유입 정도(Delta) = α × (Inflow – Outflow) + (1 - α) × Inflow
이때, α = 0.85

예측된 다음 날의 유입 정도에 따른 **Heatmap** 출력



스케줄 추천 알고리즘 설계 시 필요한 정보

- **Bike_Stop_Time :** 각 대여소 간 자동차로 걸리는 시간
  - 자동차 경로안내 API를 이용하여 **51×51개의 경로**에 대한 소요 시간을 구함
  - 소요 시간의 평균 값은 **836.969초 ⇒ 13분 56.969초 ⇒ 약 15분**
  - 각 대여소 간 소요 시간을 모두 동일하게 15분으로 가정함

In [16]:
```
temp3 = []
for k in temp2.values():
    temp3 += [k]

avg_time = np.mean(np.array(temp3))
avg_time
```

Out[16]:
```
836.9686274509804
```

In [15]:
```
print(datetime.timedelta(0, avg_time))
```

```
0:13:56.968627
```

- **top_13_sorted :** 유입 정도의 합이 큰 TOP 13 대여소의 ID
- **desired_order :** 각 대여소마다 유입 정도가 큰 순서로 (Time Slot, 유입 정도) 정렬
- **largest_stop :** 각 시간대마다 그 시간대가 제일 유입 정도가 큰 대여소의 ID
- **visit_count :** 각 대여소를 방문한 횟수

* 자동차 경로안내 API: tmap (http://tmapapi.sktelecom.com/) 제공

스케줄 추천 알고리즘

- **가정**
  - 방역에 참여하는 팀은 2개 (A팀, B팀)
  - 대여소간 소요 시간 = 15분, 대여소별 소독 시간 = 15분
    - 한 팀은 한 시간마다 2개의 대여소 소독 가능 ⇒ 두 팀은 총 4개의 대여소 소독 가능
  - 소독은 이전 Time Slot의 유입 정도를 토대로 수행함
    - ex) Time Slot 8(08~09시)에 유입 정도가 큰 대여소는 Time Slot 9(09~10시)에 소독함
    - 유입 정도 예측은 5~21시에 대해서, 방역 스케줄 추천은 6~22시에 대해서 이루어짐
  - TOP 13 대여소는 한 번 더 소독함

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']}
```

- **각 Time Slot에 소독할 대여소 배정 순서 (`visit_order`)**
  1. 직전 Time Slot에서 제일 유입 정도가 컸던 대여소들을 배정함 (최대 4개)
  2. 제일 유입 정도가 컸던 대여소의 개수가 4개가 넘는 Time Slot의 경우, 그 다음 Time Slot으로 가득 채워 배정함
  3. Time Slot 21의 배정이 끝난 경우, 방문 횟수(**`visit_count`**)가 0인 대여소와 TOP 13 대여소를 무작위로 남는 Time Slot에 배정함
  4. A팀에는 각 Time Slot의 홀수 번째 대여소, B팀에는 각 Time Slot의 짝수 번째 대여소를 배정함

효용성 검증 – 추천 스케줄과 랜덤 스케줄의 비교

- **예상 유효 소독 대수**
  - Σ {(각 대여소의 **기존 거치대수**) + (각 대여소가 소독되기 **직전 Time Slot까지의 (Inflow – Outflow) 값의 합**)} = Σ (**`disinfected[stop_id]`**)
  - 이때, **`disinfected[stop_id]`** < 0 인 경우, 0으로 처리함
  - 추천된 방역 스케줄과 랜덤으로 짜여진 방역 스케줄의 효용성을 비교하는 척도

- Case 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>

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

효용성 검증 – 추천 스케줄과 랜덤 스케줄의 비교

- Case 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>

추천 방역 스케줄

※ {'시간대': ['대여소ID-1', '대여소ID-2'], ...} 형식입니다.

> A 팀의 추천 방역 스케줄은 다음과 같습니다.

{6: ['643', '604'], 7: ['650', '651'], 8: ['646', '600'], 9: ['636', '606'], 10: ['638', '633'], 11: ['602', '628'], 12: ['663', '656'], 13: ['627', '654'], 14: ['608', '641'], 15: ['631', '609'], 16: ['625', '626'], 17: ['659', '658'], 18: ['614', '657'], 19: ['625', '657'], 20: ['631', '630'], 21: ['648', '626']}

> B 팀의 추천 방역 스케줄은 다음과 같습니다.

{6: ['609', '642'], 7: ['652', '617'], 8: ['613', '616'], 9: ['661', '660'], 10: ['605', '637'], 11: ['639', '640'], 12: ['615', '610'], 13: ['607', '623'], 14: ['621', '634'], 15: ['624', '646'], 16: ['660', '643'], 17: ['601', '622'], 18: ['647', '635'], 19: ['624', '634'], 20: ['635', '622'], 21: ['641', '612']}

예상 유효 소독 대수

약 619.4 대

619.4대

>

<랜덤 방역 스케줄>

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

효용성 검증 – 추천 스케줄과 랜덤 스케줄의 비교

- Case 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>

추천 방역 스케줄

※ {'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']}

예상 유효 소독 대수

약 600.23 대

600.23대

>

<랜덤 방역 스케줄>

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

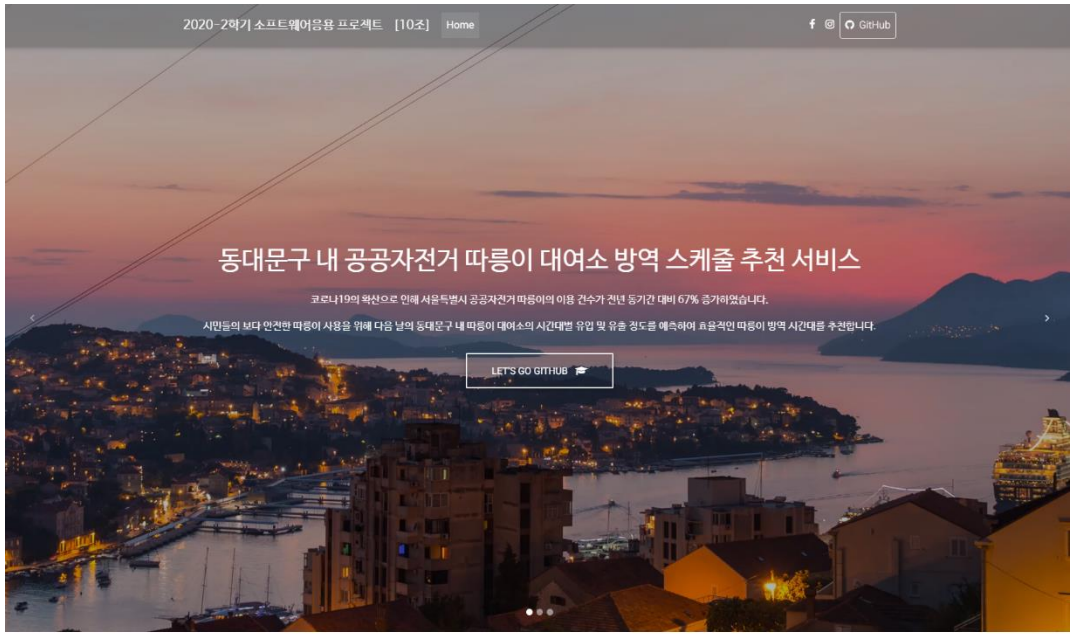
565.52대

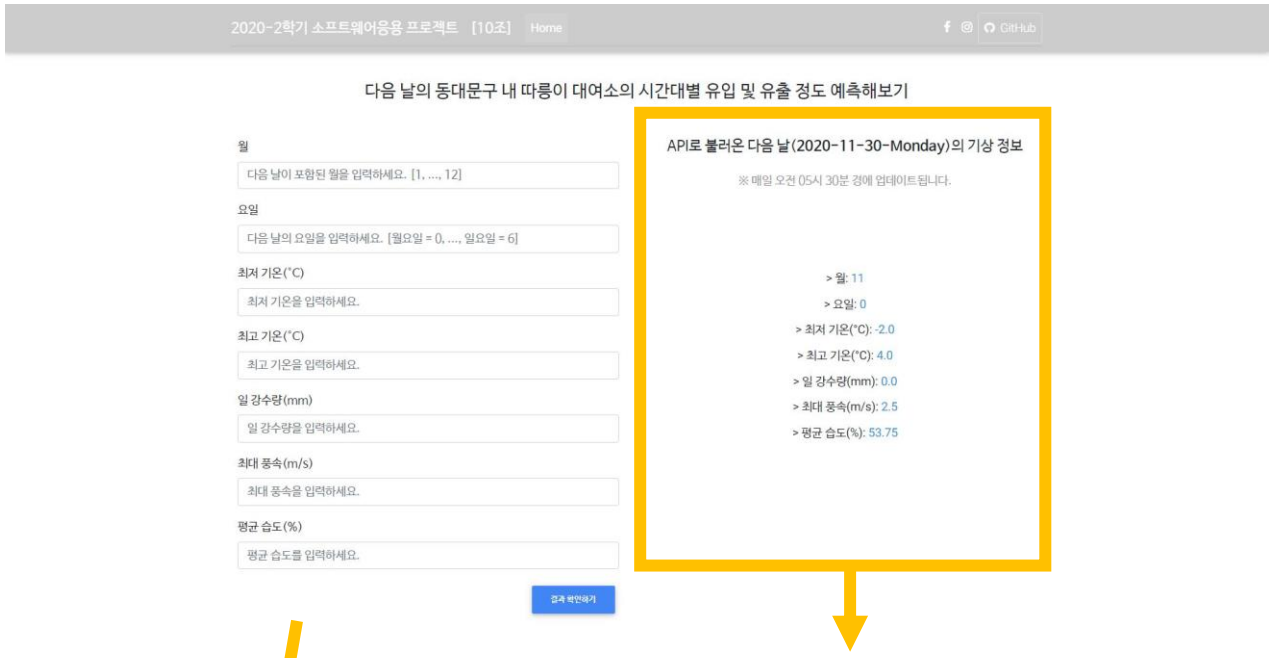
In [143]:
```
disinfected_day_ran = 0
for stop_id in Bike_Stop_ID:
    #print(disinfected_ran[str(stop_id)])
    if disinfected_ran[str(stop_id)] > 0:
        disinfected_day_ran += disinfected_ran[str(stop_id)]
print("랜덤 경로의 하루 예상 유효 소독 대수: ", disinfected_day_ran)
```

랜덤 경로의 하루 예상 유효 소독 대수: 577.5639467835426

577.56대

Flask를 활용한 웹 서비스 구축





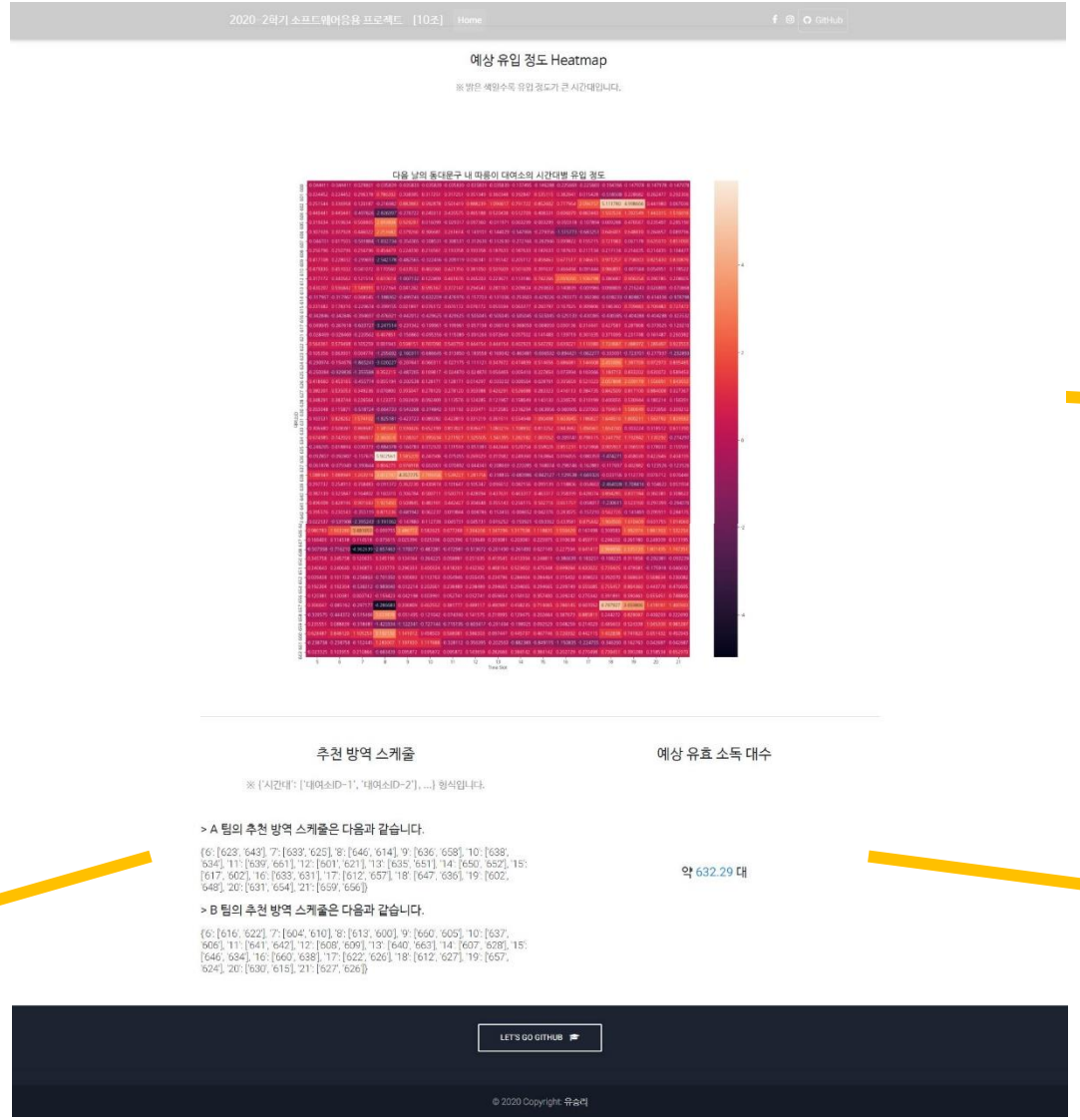
API로 불러온 다음 날의 기상 정보

파라미터 입력

(사용자 마음대로 입력 가능)

* 무료 템플릿: MDBootstrap (https://mdbootstrap.com/freebies/) 제공

Flask를 활용한 웹 서비스 구축



예상 유입 정도 Heatmap

추천 방역 스케줄

예상 유효 소독 대수

고찰 및 확장 가능성

- **파라미터를 추가**하여 **더 많은 횟수의 실험**을 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 크기**를 모두 고려하여 방역 스케줄을 도출해내도록 알고리즘을 개선한다면 대략적인 시간(hour) 단위가 아닌 **minute 단위의 스케줄 추천**이 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 부족**했을 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 모델**로도 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 적용**될 수 있을 것으로 보인다. 또한, 동대문구와 따릉이 이용량이 많은 마포구, 영등포구, 송파구 뿐만 아니라 **서울시 내 모든 구에 대해서도 확장이 가능**할 것이다.

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/

감사합니다.