광주 인공지능 사관학교

시계열 데이터 분석 및 모델개발

Ainergy

김정훈, 박선홍, 박우성, 정민우

Content

Part 1

프로젝트 주제, 목적

주제 – 공공데이터를 이용한 가스 수요 예측 모델개발(공모전)

공급사별 2013 ~ 2018 년도 시간단위의
시계열 가스 공급량 데이터를 토대로
2019년도 1, 2, 3월의 시간별 가스 공급량 예측 모델 개발.

목적 - 모델 개발을 통한 비용 절감

시간대별 도시가스 수요를 정확하게 예측 하는 것은 공급자의 비용 절감 뿐만 아니라 안정적인 배관망 관리 측면에서도 중요.
시간대별 가스 생산량이 실제 수요보다 많은 경우에는 과다한 저장 시설 운용 및 불필요한 생산 비용이 발생하는 문제가 있다.
따라서 공급량 예측 모델 개발로 불필요한 비용을 절감하고자 한다.

Part 2

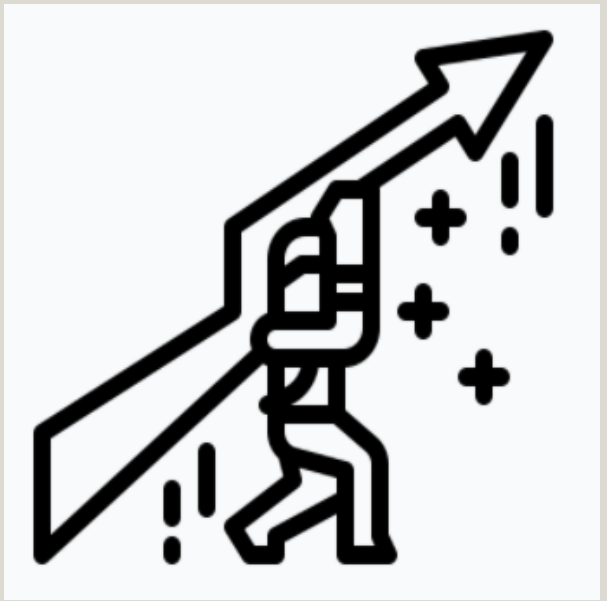
프로젝트 진행배경

통계적 기법 활용

빅데이터 분석 과목 시간에 배운 다양한 통계적 기법들은 효율성, 생산성 향상을 위해 향후 여러 산업 분야에서 적극적으로 활용될 것으로 예상.

분석모델 활용

시계열 분석에 사용 되어지는
고전적인 모델링 방법 ARIMA,
머신 러닝 XGBoost, LightGBM, CatBoost
딥러닝 RNN, 1D conv, LSTM으로
순차적으로 모델 활용



실제 데이터에 적용

실제 데이터의 EDA분석과 결측치 값 보간 등 순차적으로 데이터의 가공.
'가스 공급량 예측 모델' 제작을 통해 여러 통계적 기법과 분석모델을 활용해 보고자 한다.

Part 3

개발환경

Jupyter

로컬 환경에서 각자 데이터 분석모델 개발

Slack

Slack 채널을 통해 프로젝트 관련 소통

Github

최종 결정된 버전을 github에 업로드

Part 4

팀원 역할

박우성

- 팀장
- 데이터 전처리 시각화
- LSTM 모델링
- CatBoost 모델링

김정훈

- 데이터 전처리
  결측치 보간
- 외부데이터 크롤링
- LSTM 모델링

정민우

- LSTM 모델
- XGBoost 모델링
- 선형회귀 모델링

박선홍

- 외부데이터 크롤링
- ARIMA, 고전적
  시계열 모델링
- XGBoost 모델링
- 모델 앙상블

Part 5

프로젝트 프로세스

10 2021 October

SUN	MON	TUE	WED	THU	FRI	SAT
26 W39	27	28	29	30	1	2
3 개천절 W40	4	5	6	7	8	9 한글날
10 W41	11	12	13	14	15	16
17 W41	18 팀 일정 계획	19 팀 일정 계획	20 프로젝트 계획서 작성 Meeting	21 프로젝트 계획서 작성	22 참여할 공모전 찾기	23 참여할 공모전 찾기
24 W42	25 데이콘 시계열 분석 1등 코드 학습	26 데이콘 시계열 분석 1등 코드 학습	27 데이콘 시계열 분석 1등 코드 학습	28 Meeting(강사님) 1등 코드 같이 리뷰 공모전 참여신청 마감	29 Meeting(강사님) 1등 코드 같이 리뷰	30
31 W43	1	2	3	4	5	6

Part 5

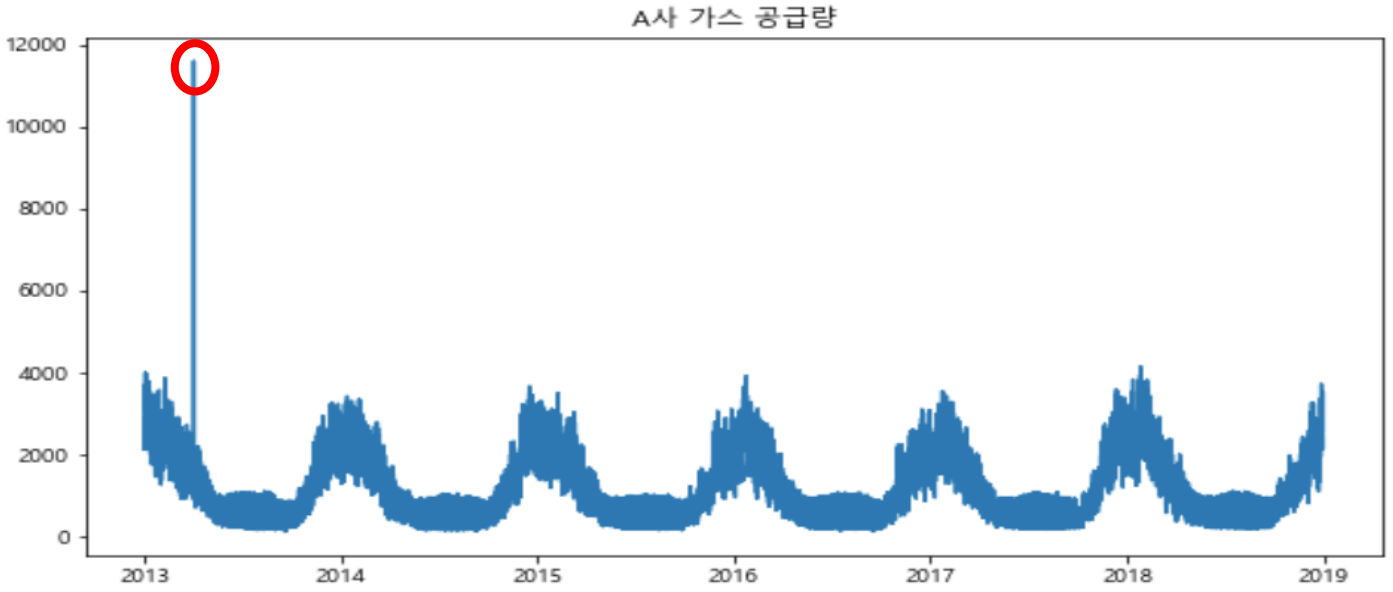
프로젝트 프로세스

11 2021 October

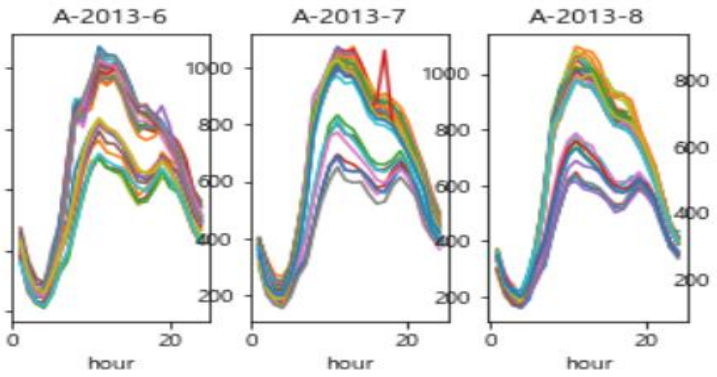
SUN	MON	TUE	WED	THU	FRI	SAT
31	1	2	3	4	5	6
	시계열 데이터 EDA분석 및 결측치 전처리					
		Meeting			Meeting	
W44						
7	8	9	10	11	12	13
	참여 대회와 유사한 논문 구현 및 비교(ARIMA)					
		호남 ICT 공모전 준비 (시계열 분석과는 무관)				
		Meeting		Meeting		
W45						
14	15	16	17	18	19	20
	호남ICT 발표 (3등 우수상 수상)	LSTM 기본 모형 적용 공부		LSTM 공모전 데이터에 적용		
		Meeting			Meeting	
W46						
21	22	23	24	25	26	27
	기준을 다르게 하여 LSTM 적용					
			태양 전력량 예측 1등 코드 xgboost 적용			
			Meeting		Meeting	
W47						
28	29	30	1	2	3	4
	프로젝트 마무리(발표영상 제작)					
W48						

Part 6

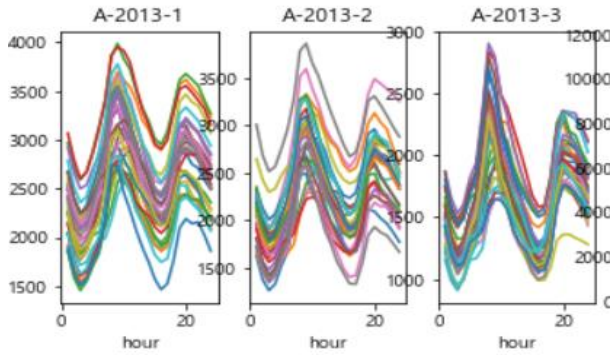
프로젝트 단계별 내용 - EDA



연간 데이터 분포



시간대별 -여름



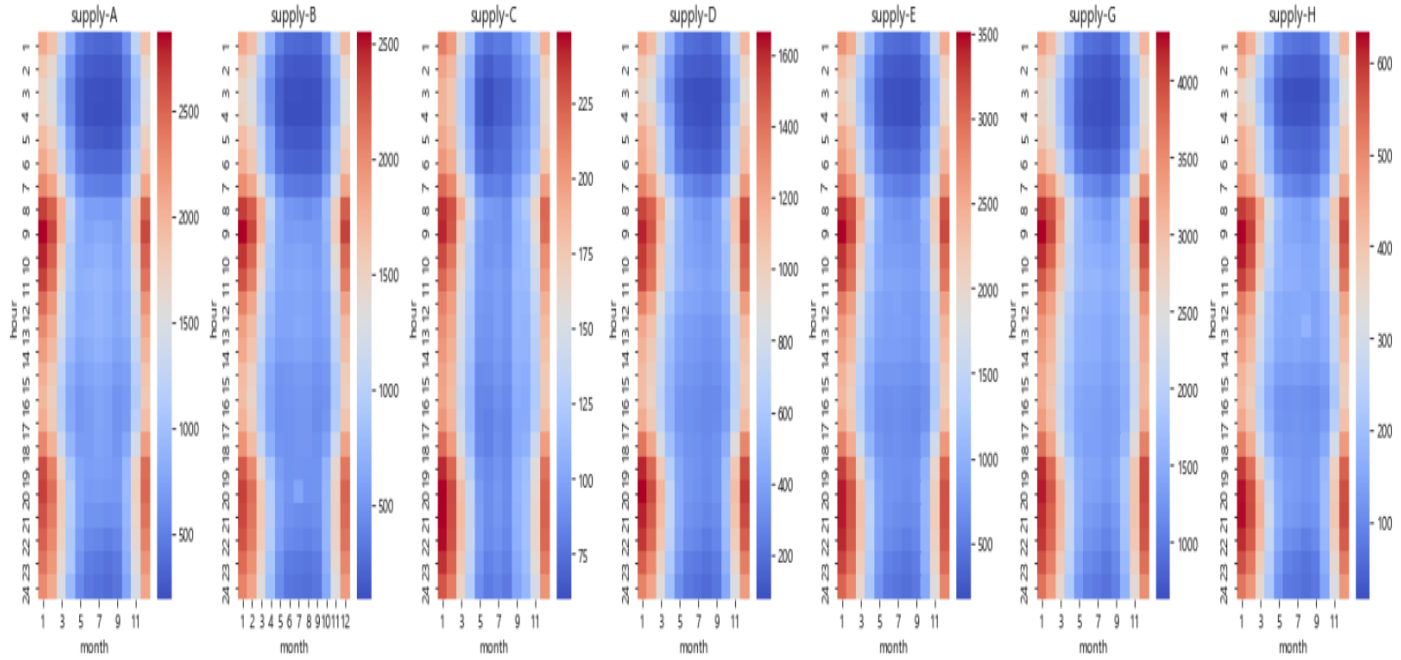
시간대별 - 겨울

EDA – 계절성 확인

- 그래프를 통해서 연간 사이클과 일간 사이클 두개를 관측 할 수 있었다.
- 연간 사이클은 겨울에는 증가하고 여름에는 감소하는 추세를 반복하고 있다.
- 일간 사이클로는 여름과 겨울의 사이클이 다르며, 여름보다 겨울의 변동폭이 더 큰 것을 확인 할 수 있다.
- 주로 추세를 잘 따라가고 있지만, 지나치게 크거나 낮은 이상치도 확인 할 수 있었다.

Part 6

프로젝트 단계별 내용 - EDA



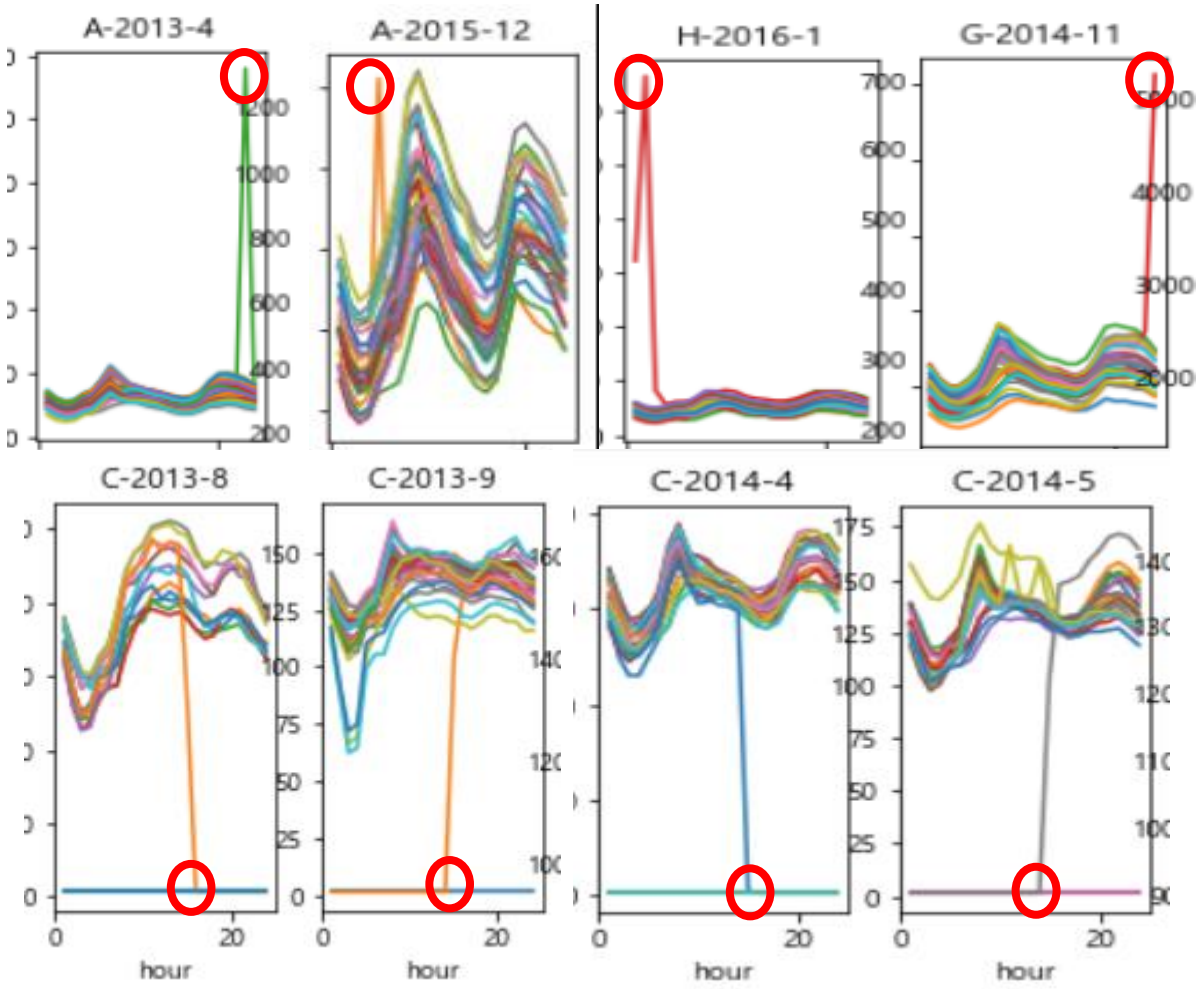
공급사별 공급량 분포

EDA – 데이터 분포 확인

- 공급사마다 가스공급량 분포가 다르다는 것을 확인할 수 있다.
  G사는 최대값이 4000이 넘어가지만 C사는 최대값이 300도 안되는 것을 확인 할 수 있다.
- 겨울인 12월, 1월, 2월은 공급량이 훨씬 많은 것을 볼 수 있고, 여름인 7,8,9월은 공급량이 적은 것을 확인 할 수 있다.
- 시간대별로는 8~11, 19~23시가 공급량이 많고, 02~05시는 적은 공급량을 나타냄.

Part 6

프로젝트 단계별 내용 - EDA



EDA – 이상치 확인

- A사는 0~ 1200의 값, H사는 0 ~ 700 사이의 값, C 사는 0 ~ 200 사이의 값을 가지고 있음을 확인 => **공급사마다 target 값이 상이함.**
- 2013-4, 2013-12 A사, 2016-1 H사, 2014-11 G사처럼 위로 **크게 튀는 이상치 값** 확인가능.
- 주로 C사에만 **0에 가까운 값을 가진 이상치** 들이 있는 것을 확인.

Part 6

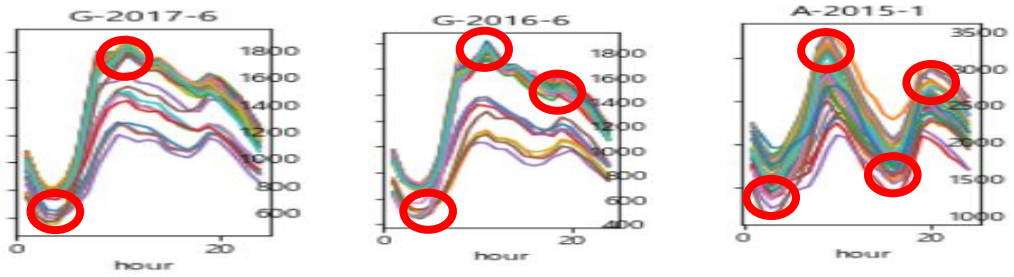
프로젝트 단계별 내용 - 전처리

```
# 일별로 이상치들을 감지하여 결측치값으로 대체해줍니다.

for supply in supplys: # 공급사별
    supply_df = total[total['supplys'] == supply]
    for year in tqdm(supply_df['year'].unique()): # 연도별
        supply_year_df = supply_df[supply_df['year'] == year]
        for month in supply_year_df['month'].unique(): # 달별
            supply_year_month_df = supply_year_df[supply_year_df['month'] == month]
            for day in supply_year_month_df['day'].unique(): # 일별
                supply_year_month_day_df = supply_year_month_df[supply_year_month_df['day'] == day]
                Q3 = supply_year_month_day_df.describe()['target'].loc['75%']
                Q1 = supply_year_month_day_df.describe()['target'].loc['25%']
                IQR = Q3 - Q1
                Q3_outlier = Q3 + 1.6*IQR  # Q3의 이상치 처리
                Q3_outlier_df = supply_year_month_day_df[supply_year_month_day_df['target'] > Q3_outlier]

                if not Q3_outlier_df.empty: # df에 Q3_outlier보다 큰값이 존재하여 빈 데이터프레임이 아닐때
                    # 전체 df total에 가져온 index의 자리를 Nan값으로 처리해줍니다.
                    total.loc[Q3_outlier_df.index, 'target'] = np.nan

total['target'].interpolate(method='polynomial', order=3, inplace=True)
```



3차 4차 5차

이상치 처리 – 결측치 생성, 보간

- 위로 크게 튀고 있는 이상치 처리.
- 'target' 값이 Q3 + 1.6 * IQR 값보다 더 크면 이상치라 생각하고 결측치로 변환.
- 그래프를 보면 3,4,5차 함수의 개형을 따르고 있는 것을 볼 수 있음.
- 판다스의 interpolate 메서드 중 'polynomial', order = 3를 사용해서 삼차함수 모양으로 결측치를 보간함

Part 6

프로젝트 단계별 내용 - 전처리

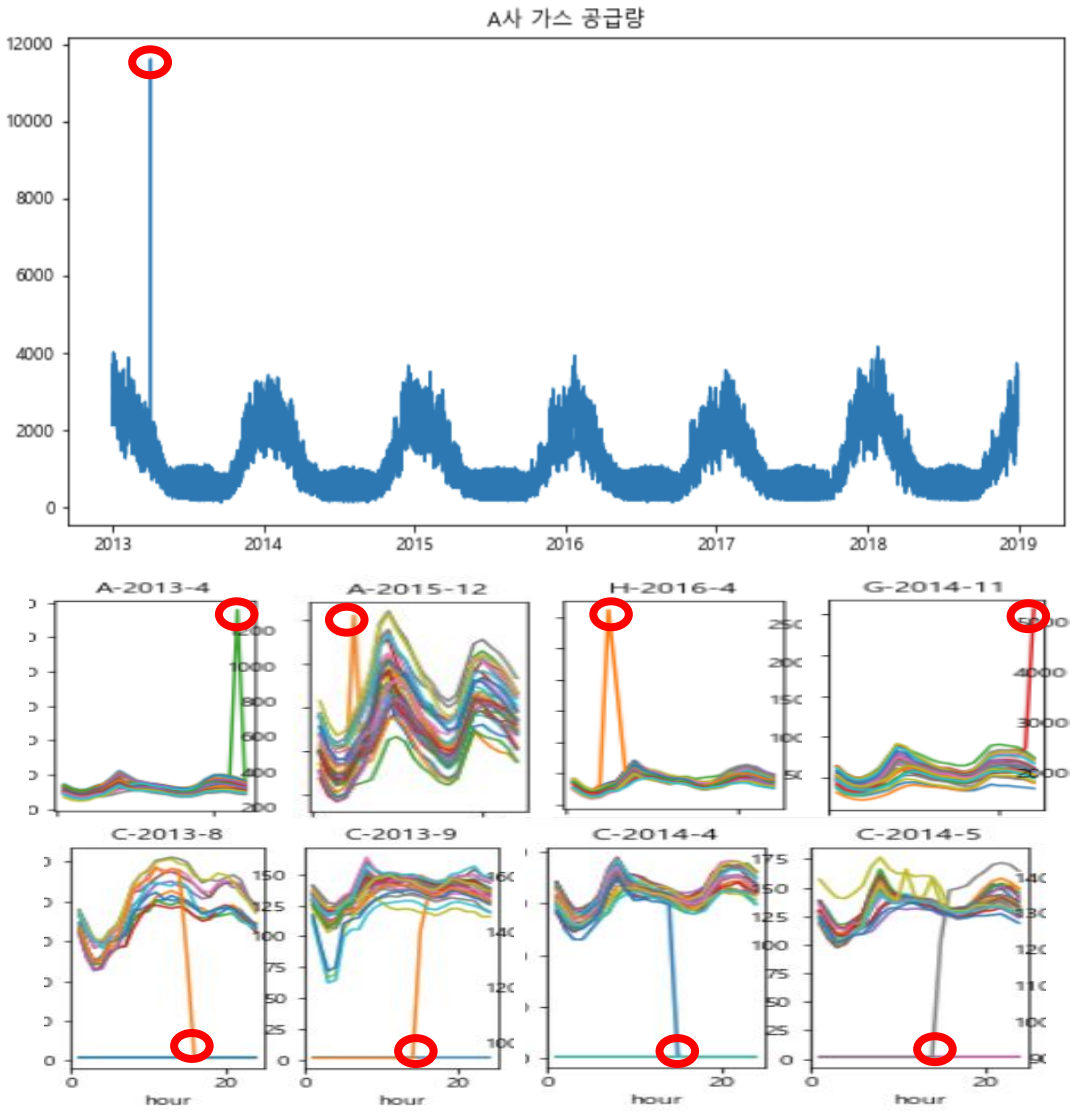
```
# nan값으로 바뀔 index 저장
nan_index_list = []
for month in month_list:
    C_month = C[C['month'] == month]
    hour_list = list(range(1, 25, 1))
    # 같은 month, 같은 시간대의 IQR범위 구하기 위해서이다.
    for hour in hour_list:
        C_hour = C_month[C_month['hour'] == hour]
        Q1 = C_hour['target'].quantile(0.25)
        Q3 = C_hour['target'].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5*IQR
        # 값이 lower_bound 이상치보다 작으면서 99미만이거나, 1.378, 2.756 중 하나이면 이상치로 간주
        condition = C_hour[(C_hour['target'] < lower_bound) & (C_hour['target'] < 100) | (C_hour['target'].isin([1.378, 2.756])) ]
        nan_index_list += condition.index.tolist()
# 인덱스에 있는 부분들을 모두 결측치로 반환
for i in nan_index_list:
    total.loc[i,'target'] = np.nan
# total 데이터 프레임에서의 인덱스를 가져옵니다.
C_index = total[total['supplys'] == 'C'].index.values
# 해당 인덱스의 target 데이터를 C_total_target_list로 바꾸어줍니다.
total.loc[C_index, 'target'] = C_total_target_list
C_target_T.reset_index(inplace= True)
C_target_T = C_target_T.interpolate(method='values')
# 맨 앞에 있는 2013 값은 바로 채워지지 않기 때문에 2014 공급량 값을 가지고 채운다.
C_target_T = C_target_T.fillna(method='bfill')
# 다시 전치
C_target = C_target_T.transpose()
# 인덱스 넣기
C_target = C_target.rename(columns = C_target.iloc[0])
C_target.drop(C_target.index[0],inplace= True)
```

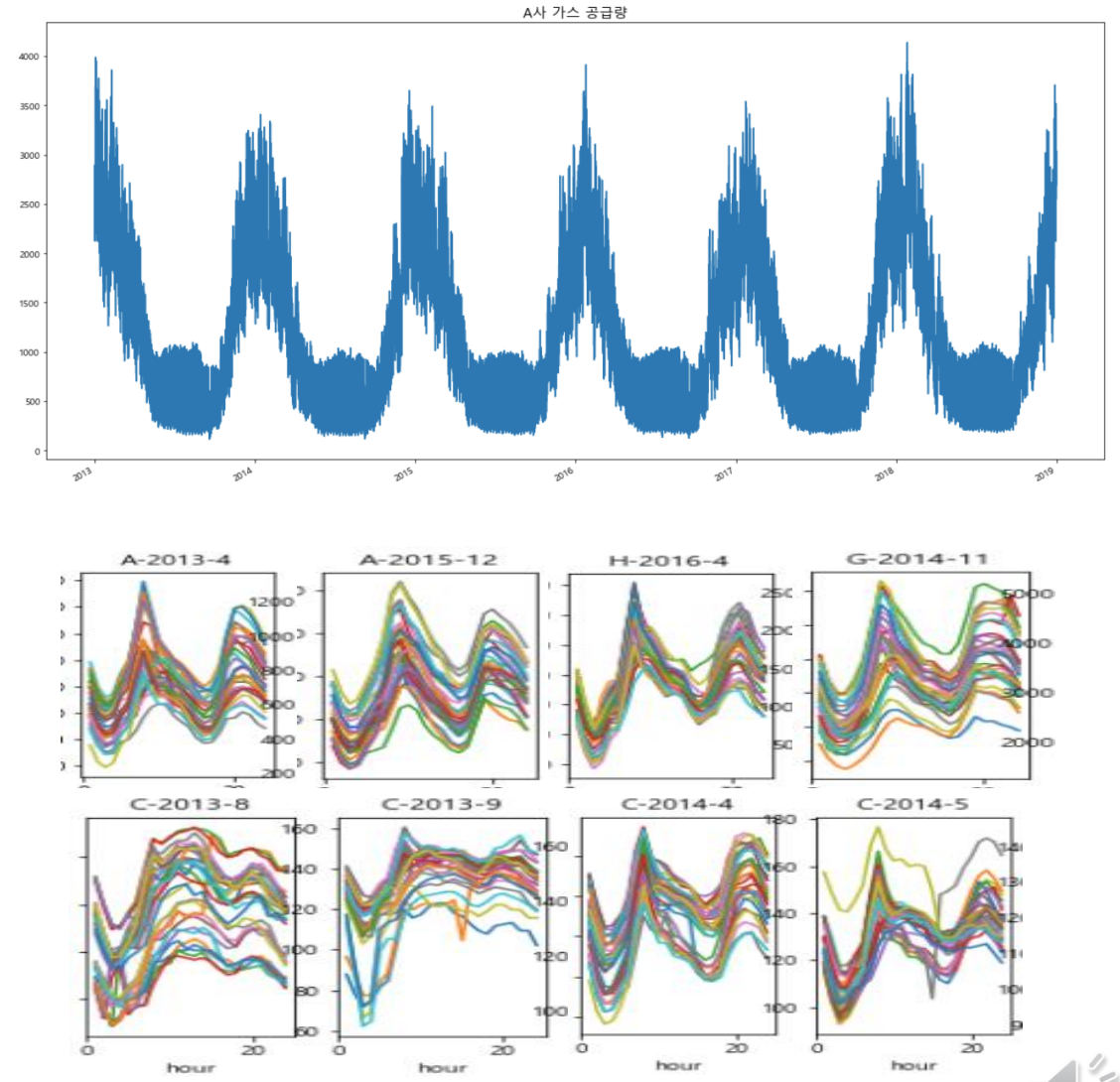
이상치 처리 – 결측치 생성, 보간

- 0에 가까운 이상치들 처리
- Q3 - 1.5 * IQR(Q3-Q1) 값보다 더 작은 값을 이상치라고 설정, 'target'의 값이 이상치보다 더 작거나 0에 가까운 값인 2.756, 1.378이면 결측치로 변환.
- 2014년 데이터가 결측치면 이 값은 2013년 2015년의 값에 영향을 받을 것이라고 생각.
- 선형 보간법을 사용하기 위해 method = 'values' 사용.

Part 6

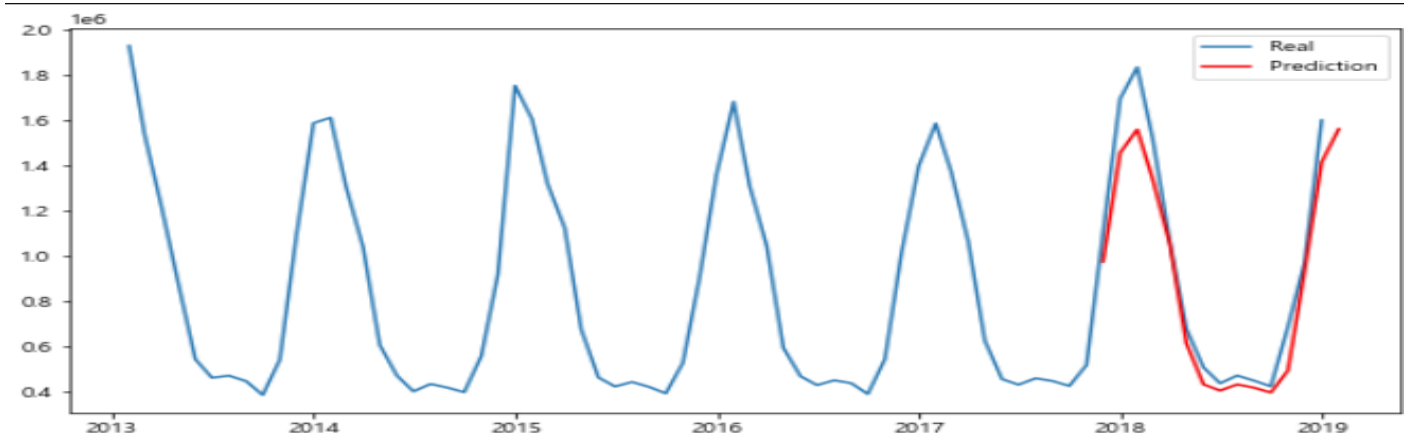
프로젝트 단계별 내용 – 전처리(결과)

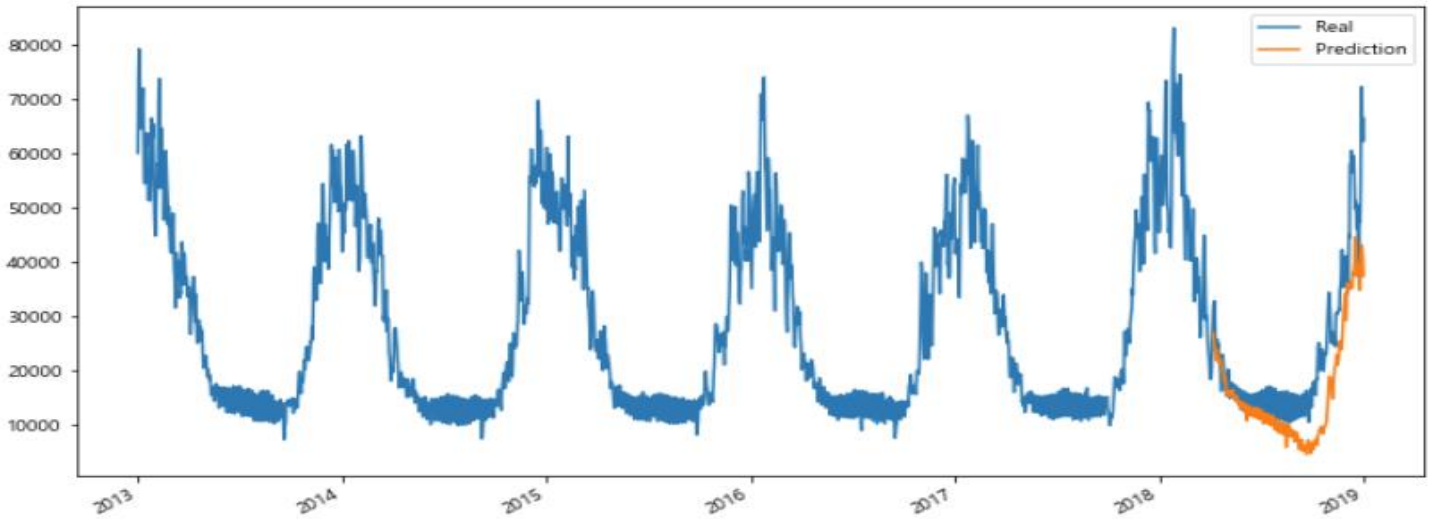




Part 6

프로젝트 단계별 내용 – 고전적 시계열 모델





ARIMA & 지수평활법

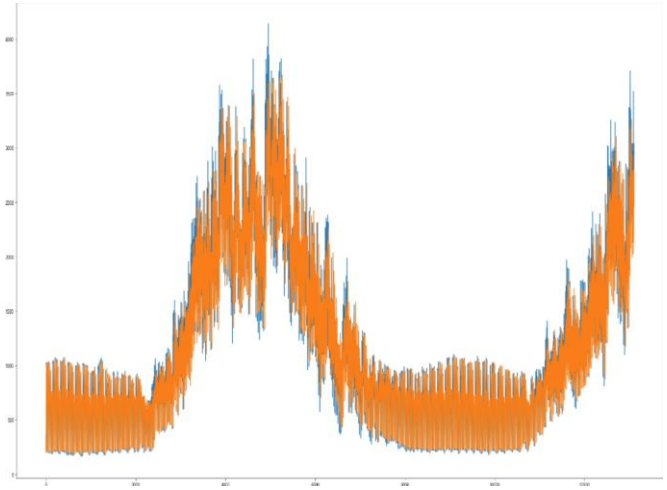
- (위) 월 데이터로 **ARIMA 모델** 구현 *(MAPE = 10.33)*
- (아래) 일일 데이터 **Holt-Winters 가법모형** *(MAPE=30.48)*
- 시간단위로 예측하지 않은 이유는 **비정상적인 구동 시간 및 큰 오차**(AIC > 40,000)
- 전통적 시계열 분석 방법으로는 시간단위로 예측해야 하는 것이 사실상 어려움

Part 6

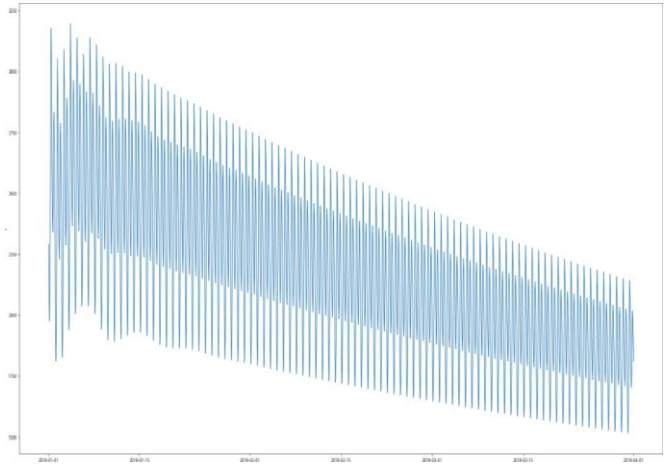
프로젝트 단계별 내용 - 회귀모형

회귀모형

T24_168: $Y_t = \beta_{24} X_{t-24} + \beta_{168} X_{t-168} + \beta_0$



Test Data



2019년 1~3월 예측

논문 Review

- 시간대별 가스 예측 주제 논문에서 Insight를 얻기 위해 하루 전 공급량과 일주일 전 공급량을 변수로 한 다변량 회귀분석을 진행해 봄

- Test Data에서는 비교적 정확한 수치가 나왔으나*(R2 score = 95.99, MAPE = 7.14)* 예측한 값을 변수로 넣었을 때(장기간 미래를 예측해야 할 때)에는 유효하지 않음

- **최대 24시간** 까지만 예측할 수 있어 모델을 그대로 사용할 수는 없었으나, 이후 회귀분석에서 시간대별, 요일별로 공급량을 활용한 통계값들을 사용하는 계기가 됨

Part 6 프로젝트 단계별 내용 - XGBoost

0.1.1. 기본적인 변수 추가(날짜, 월 등)

```
A_df_var = A_df.reset_index()

# 날짜, 시간 변수 추가
A_df_var['hour'] = A_df_var['index'].dt.hour
A_df_var['weekday'] = A_df_var['index'].dt.weekday
A_df_var['month'] = A_df_var['index'].dt.month
A_df_var['week'] =  A_df_var['index'].dt.weekofyear
```

```
from tqdm import tqdm

# 요일별, 시간별 평균, 표준편차를 dataframe 값 원소 조건에 따라 붙여준다
def to_mean_std(df):

    # progress bar 표시하기 위해 사용
    tqdm.pandas()

    power_week_hour_mean = pd.pivot_table(df, values = 'target', index = ['hour', 'weekday'], aggfunc = np.mean).reset_index()
    df['week_hour_mean'] = df.progress_apply(lambda x : power_week_hour_mean.loc[(power_week_hour_mean.hour == x['hour']) & (pow

    power_hour_mean =  pd.pivot_table(df, values = 'target', index = ['hour'], aggfunc = np.mean).reset_index()
    df['hour_mean'] = df.progress_apply(lambda x : power_hour_mean.loc[(power_hour_mean.hour == x['hour'])  ,'target'].values[0]

    power_week_hour_std = pd.pivot_table(df, values = 'target', index = ['hour', 'weekday'], aggfunc = np.std).reset_index()
    df['week_hour_std'] = df.progress_apply(lambda x : power_week_hour_std.loc[(power_week_hour_std.hour == x['hour']) & (power_

    power_hour_std = pd.pivot_table(df, values = 'target', index = ['hour'], aggfunc = np.std).reset_index()
    df['hour_std'] = df.progress_apply(lambda x : power_hour_std.loc[(power_hour_std.hour == x['hour'])  ,'target'].values[0], a

    return [power_week_hour_mean, power_hour_mean, power_week_hour_std, power_hour_std], df
```

변수 탐색

- XGBoost로 회귀분석을 한 이유는 요일, 시간 등의 시간 변수들을 설명변수에 입력함에 따라 시계열 데이터 처리도 가능하기 때문
- 우선 시계열 데이터 분석을 위해 **시간, 요일, 주, 월 데이터**를 넣음
- 앞의 논문 및 과거 DACON 전력사용 예측 모델 순위권 모델을 참고하여 요일 및 시간별 trend를 반영하기 위해 **요일, 시간대별 공급량 평균 및 표준편차**를 넣음.

Part 7

프로젝트 단계별 내용 - XGBoost

0.1.2. cyclical endocing(시간)

```
# cyclical endocing(시간)
# 시간은 0~24시가 마치 삼각함수처럼 cycle이 반복되기 때문에 다음과 같이 인코딩

A_df_var['sin_time'] = np.sin(2*np.pi*A_df_var.hour/24)
A_df_var['cos_time'] = np.cos(2*np.pi*A_df_var.hour/24)
```

```
# train, test = A_df_reg_final.iloc[:, 1:],  A_df_reg_final.iloc[:, 0]
X_train, X_test, y_train, y_test = train_test_split(train, test, random_state=2019, test_size=.25)
pipe.fit(X_train, y_train)
```

```
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('base_model',
                 XGBRegressor(base_score=0.5, booster='gbtree',
                              colsample_bylevel=1, colsample_bynode=1,
                              colsample_bytree=1, enable_categorical=False,
                              gamma=0, gpu_id=-1, importance_type=None,
                              interaction_constraints='',
                              learning_rate=0.300000012, max_delta_step=0,
                              max_depth=6, min_child_weight=1, missing=nan,
                              monotone_constraints='()', n_estimators=250,
                              n_jobs=12, num_parallel_tree=1, predictor='auto',
                              random_state=0, reg_alpha=0, reg_lambda=1,
                              scale_pos_weight=1, subsample=1,
                              tree_method='exact', validate_parameters=1,
                              verbosity=None))])
```

```
pred_xgb = pipe.predict(X_test)
```

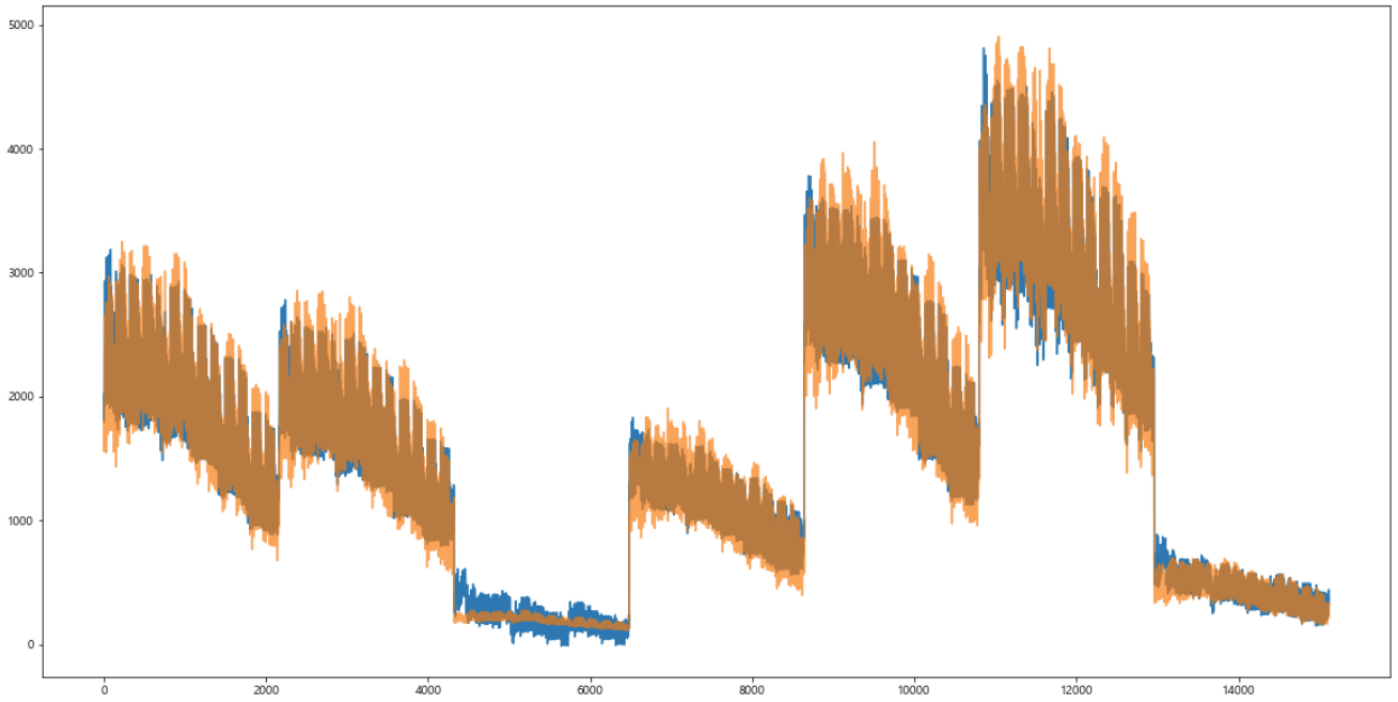
A사 대상 test

- 그리고 0~24시까지 계속 주기적으로 반복하는 시간 특성상 hour을 삼각함수 형태로 변형해서 설명변수에 넣어 줌 (cyclical encoding)

- 시간별 데이터로 A사를 대상으로 했을 때 준수한 성능을 보여줘 전체 모델에 test *(MAPE=11.35)*

Part 6

프로젝트 단계별 내용 - XGBoost



XGBoost 결론

- 7개 공급사에 대한 각각의 모델을 만들어 GridsearchCV 등을 활용해 최적의 parameter을 찾고 훈련시킴 (MAPE = 11.57)
- 훈련시킨 모델을 통해 2019년 1~3월 총 15,120개의 데이터를 예측함
- 파란색이 baseline이고 밝은 색이 custom model인데, baseline에 비해 C사에서 예측이 많이 차이가 나는 것을 볼 수 있다.
- 실제 DACON public 에서도 0.1177(MAPE 기준 11.77)을 기록해 **전처리 효과 및 모델 개선 효과도 볼 수 있었음**

Part 6

프로젝트 단계별 내용 - LSTM

```python
# 달별 전날 12시간 과 그 다음날 24시간을 하나의 데이터(36시간)로 만들어 그달의 첫날과 마지막날을 제외한 데이터 갯수만큼 생성되는 함수
def make_36_data(df, supply, year, month):
    df_supply = df[df['supplys'] == supply] # 공급사 선택
    df_supply_year = df_supply[df_supply['year'] == year] # 년도 선택
    df_supply_year_month = df_supply_year[df_supply_year['month'] == month].reset_index(drop=True) # 달별 선택


    # 예시) 2013년은 1월 1일 전의 5시간 데이터가 없어서 건너띄어줌(1월 2일부터)
    # 1월 1일 12시 ~ 1월 2일 23시 (36시간 학습)
    # 1월 1일 12시 ~ 1월 31일 23시 총 29개 생성

    start_index = 12 # index 위치 1월 12시
    end_index = 48 # index 위치 1월 12시 +36 -> 1월 2일 23시
    row_lenght = end_index - start_index
    day = df_supply_year_month['day'].max() - 2 # 총 데이터의 갯수 전날이 없는 첫번째 데이터, 마지막 그 다음날이 없는 데이터를 빼주어 -2

    data = []
    for i in range(day):
        start_index += 24  # 인덱스 위치를 하루씩 밀어준다.
        end_index += 24
        one_picture = df_supply_year_month[start_index:end_index]['target'].to_list() # 데이터의 수를 사진이라 생각하여 리스트에 추가
        data.append(one_picture)

    data = np.array(data).reshape(day,row_lenght,1) # 마지막으로 데이테수, 데이터의 길이, target으로 생성.
    return data
```

LSTM 학습 데이터 생성

- 공급사별, 년도별, 월별에 대해서 전날 12시간의 데이터로 다음날 24시간을 예측하기 위해 시계열 데이터의 길이를 36시간으로 만듦.
- 달의 시작일은 그 전날의 12시간이 없기에 제외하며 또한, 마지막날에는 다음날 24시간이 없기에 제외.
  (한달이 총 31일 경우 29일의 데이터 생성)
- 공급사별 2013 ~ 2018 년 해당 달에 대해서만 데이터들을 병합
  1, 3월은 총 174개 (29 × 6)
  2월은 156개(26 × 6) 데이터 수를 가진다 .

Part 6

프로젝트 단계별 내용 - LSTM

```
def train_model(train_data):
    # return_sequences = True: "Sequence to Sequence" 모델을 의미한다.
    # input_shape = (None, n_input) :  "None"은 가변적인 시계열 길이를 나타낸다.
    my_model = Sequential()
    my_model.add(LSTM(units=64,return_sequences=True, input_shape=(None, 1),activation="tanh"))        # LSTM.
    my_model.add(TimeDistributed(Dense(units=1)))    # Wrapper.

    # Optimizer 객체 생성 후 컴파일한다.
    my_optimizer=Adam(lr=0.0001)
    my_model.compile(loss = "mse", optimizer = my_optimizer, metrics=["mse"])

    # No validation.
    # 주의: y 는 X를 +1 shifting 한 것이다
    my_EarlyStop = EarlyStopping(monitor='mse', mode='min', verbose=1, patience=10)          # 조기 종료 허락!
    my_model.fit(train_data[:,:-1,:], train_data[:,1:,:], epochs=2000, batch_size = 16, verbose = 1, callbacks=[my_EarlyStop])

    return my_model
```

LSTM 학습 모델 생성

- 월별 데이터를 제공하면 해당 월 모델을 생성하기위해 함수 작성.
- input_shape 의 None 값은 예측한 값을 다시 넣어 예측을 진행해야 함으로 가변적인 시계열이 계속 들어옴으로 None으로 설정.
- LSTM의 1층으로도 충분하며 units은 64일 때 적절하였다.
- return_squences= True 중간 스텝의 출력을 모두 사용 의미.
- TimeDistributed 각 스텝마다 cost를 계산해서 하위 스텝으로 오류를 전파하여 각 weight를 업데이트.
- Optimizer은 Adam이며 leaning rate=0.0001, 발산을 줄이며 극소점으로 잘 가기 위해 설정.
- Epochs = 2000, EarlyStopping으로 작은 learning rate를 최대한 이용하여 극소점을 찾아 멈춤. (대략 1000 ± 100 에서 멈춤)
- 들어오는 데이터의 수가 174, 156임으로 Batch_size = 16으로 설정.

Part 7 프로젝트 단계별 내용 - LSTM

```python
def model_36_predict(my_model, first_seed, days):
    n_predict_time_steps = 24                    # 예측할 길이 (24시간)
    predict_seed_steps = 12                      # 예측한 값에서 다음 예측때 사용되어질 길이 (12시간)

    data_dict = {} # 예측한 하루 24간 데이터 딕셔너리로 저장

    # 1월 2일 예측 ( 1월 1일 18시 ~ 1월 1일 23시)
    for i in range(n_predict_time_steps):
        X = first_seed.reshape(1,-1,1)        # Reshape.
        y_pred = my_model.predict(X)
        y_last= y_pred[0,-1,0]              # 마지막 출력이 바로 y.
        # 예측되어지는 24시간이 씨드값에 계속 붙여져 예측
        first_seed = np.concatenate((first_seed, np.array([y_last]).reshape(-1,1)), axis=0)

    data_dict[1] = first_seed # 1월 2일
    total = data_dict[1][predict_seed_steps:] # 나중에 데이터들을 다 붙이기 위해 만든 변수

    for day in range(1,days):
        predict_seed_slice = data_dict[day][-predict_seed_steps:] # 예측되어진 값을 들고와 다시 씨드로 설정.

        for i in range(n_predict_time_steps):
            X = predict_seed_slice.reshape(1,-1,1)        # Reshape.
            y_pred = my_model.predict(X)        # 예측
            y_last= y_pred[0,-1,0]              # 마지막 출력이 바로 y.S
            # 예측되어지는 24시간이 씨드값에 계속 붙여져 예측
            predict_seed_slice = np.concatenate((predict_seed_slice, np.array([y_last]).reshape(-1,1)), axis=0)

        tomorrow = day +1
        data_dict[tomorrow] = predict_seed_slice # 전날 12시간의 씨드 값으로 다음날 24시간을 예측값을 딕셔너리로 저장

        total= np.concatenate((total, data_dict[day][predict_seed_steps:])) # 총 31일 시간단위로 예측한 값을 한줄로 나타냄.

    return total
```

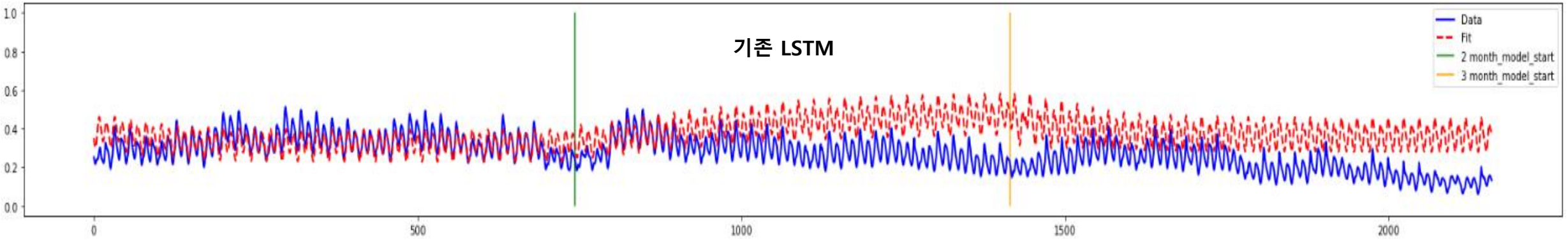
LSTM 학습 모델 예측

- 해당 월 모델, 첫 seed 12시간의 값, 예측할 일을 제공하면 예측이 진행되도록 함수 설정.
- N_predict_time_steps : 예측할 길이 24시간
- predict_seed_steps : 사용되는 seed의 길이 12시간
- 전달 마지막 일의 12시간으로 다음달 첫날 값을 예측하여 딕셔너리로 첫날의 기준을 잡아준다.
- 그 후 예측되어진 첫날 딕셔너리의 값의 12시간을 seed 값으로 설정하여 다음날 24시간을 예측한다.
- 예측 되어질 때마다 seed값을 뺀 값들(24시간)을 계속 붙여주어 총 시간별로 한달을 예측한 값을 구할 수 있다.

Part 6 # 프로젝트 단계별 내용 - LSTM

LSTM 학습 모델 결과 – (기존 LSTM)

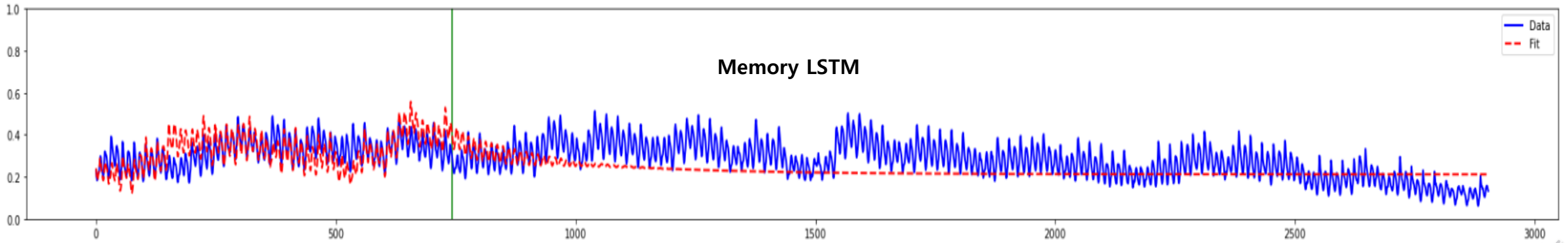
- 2018년 마지막일의 12시간을 첫 seed로 24시간을 예측하며, 또한 결과값의 12시간을 이용하여 다음 값을 예측 (월 일수만큼 반복)
- 1월 예측이 끝난 후 1월의 마지막일의 12시간을 seed로 똑같이 진행하여 2월 예측 같은 방법으로 3월 예측
- 무작위로 설정한 2014년도 1, 2, 3월 데이터를 기준으로 비교.
- 해당 월별 모델로 바꾸어 질 때 예측의 변동성이 커지며 감소하는 추세를 읽지 못하여 좋은 성능은 낼 수 없었다.



프로젝트 단계별 내용 - LSTM

LSTM 학습 모델 결과 – (Memory LSTM)

- X로는 31일치 데이터 (744), Y로는 32일의(24)을 하나의 데이터로 하며, 다음 데이터로는 X는 2 ~ 32일, Y는 33일로 계속 1일 씩 밀어 주어 데이터셋을 구한다.
  (1일 씩 밀어주는 데이터의 기간은 12월에서 4월까지 한정한다.)
- return_squences= False 로 중간 스텝의 출력 필요 없으며, TimeDistributed도 사용하지 않는다. (Many-to-one)
- 다음으로 LSTM 기존 모형과 같이 예측되어진 값을 이용하여 다음 값을 예측한다.
- 추세를 저장하게끔 설정한 LSTM에서는 예측 되어 진지 얼마 되지 않아 예측이 안되는 것을 볼 수가 있다. (시계열을 길이가 길어 예측이 어려움)



Part 7

프로젝트 결과

최종 결과

다양한 분석모델을 통해 시간별 가스 공급량 예측모델을 개발한 결과,

- **ARIMA 모델**: 비정상 시계열을 차분한 이후 활용.
  월 단위 데이터까지는 잘 예측 했지만(MAPE =10.33) 보다 세밀한 데이터는 예측 불가

- **지수평활법**: 계절지수의 평활화를 위해 holt-winters 가법모형을 적용
  일단위로 예측하였을 때 예측결과 부정확(MAPE = 30.48)

- **XGBoost**: '시간, 요일, 주, 월' 데이터와 '요일, 시간별 평균 및 표준편차'를 변수로 설정.
  공급사 별로 예측 모델을 따로 만들었는데 C사의 경우 실제와 예측 사이에 어느정도 차이가 났지만
  시간단위로 예측임을 감안하면, 전체적으로 준수한 예측성능을 보임(MAPE = 11.77)

- **LSTM**: 첫 seed 값과 예측할 길이를 한 데이터로 간주하여 학습시킬 데이터 모양을 만들어 줌.
  LSTM은 기존의 RNN 모델에 비해 장기간 예측에 효과적
  해당 프로젝트에서는 RNN보다 많이 예측 되지만 1달 이후 부터의 예측의 정확도가 많이 떨어짐(성능 측정하기 어려움)

Part 7

프로젝트 결과

프로젝트 자체 평가

- 고전적 방법의 시계열 분석으로는 시간단위의 공급량을 예측하는 데에 한계가 있었음
- 모든 예측 모델에서, 1~3월의 공급량 예측을 위해 동계 데이터만 선별하여 학습하였을 때 예측 성능이 향상됨
- 회귀모델은 변수선정과 데이터 선별에 따라 예측성능이 달라지기 때문에 통계적 직관력이 중요

기대효과

- 가스는 전력과 다르게 공급과정이 수 시간이 소요되기 때문에
  시간단위별 예측을 통해 얻는 경제적 효과도 상대적으로 높을 것으로 예상.
- 가스 배관망의 압력이 증가하면 사고 위험도 증가하기 때문에, 안정적 공급을 위해서도 정확한 수요 예측은 필요.

개선방안

- LSTM 모델에 1차원 합성곱 신경망(Conv1D)을 추가하여 입력한 시계열 데이터를 평활화
- 예측할 변수의 과거 값 뿐만 아니라 그 변수와 의존성 있는 다른 변수들의 분석도 고려한 VAR(벡터 자동 회귀 분석)
- 회귀분석에 기온 변수를 추가
- 관측이 불가능한 외란이 포함된 시계열의 예측 정확도를 향상시켜주는 칼만필터를 적용

Part 8 프로젝트 Epilogue

박우성

데이터를 시각화와 EDA를 하며 모델링을 들어가기 전 일정한 패턴을 보여 예측을 잘 할 수 있을 거라 기대 하였다. 하지만 시계열의 고전적인 모델부터 딥러닝 까지 사용 해본 결과 시간단위로 예측을 해야함으로 시계열을 길이가 길어 예측하는데 어려움이 많았다. 길이가 긴 시계열 데이터를 예측할 때에 다운 샘플링 후 다양한 방법으로 예측이 필요할 듯 하다.

김정훈

정규기간에 배우지 않았던 시계열 데이터를 다루며,고전적인 시계열 모델부터 현대 딥러닝 모델까지 사용했다. 모델을 학습시키면서 데이터 전처리의 중요성, 외부데이터를 가져올 때도 data leakage의 위험에 대해 경험하며, 결과를 확인하며 시계열 예측의 어려움을 알 수 있었다.

정민우

시계열 예측에 사용되는 여러 모델들의 장단점을 파악할 수 있었다.
시간을 반영한 예측모델을 만들었기 때문에 일반적인 주기는 잘 반영되었지만, 공급량이 급변하는 상황을 예측하기 위해선 추가적인 외부데이터가 필요하다고 느꼈다.

박선홍

교과서적인 범주를 벗어나 실제 대규모 데이터로 시계열 데이터를 다루는 것은 처음이어서 많은 시행착오가 있었지만 분석, 전처리 방법 등 많은 것을 배울 수 있었다. 팀원들 및 강사님과 재미있게 프로젝트를 진행 히면서 데이터 분석의 첫걸음을 잘 뗄 수 있었다.

Part 9

관련논문 및 Reference

- Han, J.-H., & Lee, G.-C. (2016, February 29). Forecasting Hourly Demand of City Gas in Korea. *Journal of the Korea Academia-Industrial cooperation Society*. The Korea Academia-Industrial Cooperation Society. (https://doi.org/10.5762/kais.2016.17.2.87)

- 동준이, [Analytics]동준이, 전력사용 요인 분석 및 모델링, DACON (https://dacon.io/codeshare/2743?dtype=recent)

- DACON 1등 솔루션, 4장 상점 매출 예측 (https://github.com/wikibook/dacon)