23-1 ESC Final Project

데이터 분석 프로젝트 : 제조업 에너지 소비 최적화

ESC 5조

분석개요 및 목적

Step1. Prediction

Step2. Optimization

“ 생산계획 최적화 “

Prediction

Input 변수

변수의 목적 에 따라 분류 후 일부 선정

Prediction	시간 변수: 날짜, 시간, 15분, 30분, 45분, 60분, 평균, day, d, m 날씨 변수: 기온, 풍속, 습도, 강수량 전기요금 (계절)
Optimization	공장인원, 인건비, 생산량, 전기요금

Input 변수 재정의

강수량

- 0, 0.1, ... → **강수 여부로 이진 변수화**

Day, D, M

- 1, 11, 1 (1월 11일 월요일) → **주중/주말 분할 + 공휴일 추가**

전기 요금

- 계절에 따른 이산 변수 → **카테고리화 한 후 one hot encoding**

인건비

- 1.0 / 1.5 → **평일 주간 1 / 평일 야간, 주말 주간 1.5 / 주말 야간 2**

최종 Input data

- 시간, 기온, 풍속, 습도, 강수여부, 전기요금(원 핫 인코딩), weekend(주말여부), holiday(공휴일여부)
- 기온, 풍속, 습도 칼럼에 대해 MinMaxScaler 로 스케일링 진행
- 1월부터 8월까지 학습한 후 9월 데이터 예측

	시간	기온	풍속	습도	강수여부	전기요금(계절)_109.8	전기요금(계절)_167.2	weekend	holiday
0	0	0.193833	0.315789	0.700000	0.0	1	0	0.0	1.0
1	1	0.165198	0.197368	0.766667	0.0	1	0	0.0	1.0
2	2	0.178414	0.342105	0.555556	0.0	1	0	0.0	1.0
3	3	0.174009	0.342105	0.533333	0.0	1	0	0.0	1.0
4	4	0.162996	0.342105	0.577778	0.0	1	0	0.0	1.0
...	...	...	...	...	...	...	...	...	...
6163	19	0.742291	0.473684	0.855556	1.0	0	1	0.0	0.0
6164	20	0.753304	0.552632	0.777778	1.0	0	1	0.0	0.0
6165	21	0.753304	0.565789	0.755556	1.0	0	1	0.0	0.0
6166	22	0.748899	0.328947	0.788889	1.0	0	1	0.0	0.0
6167	23	0.748899	0.328947	0.788889	1.0	0	1	0.0	0.0

6168 rows × 9 columns

모델링 with Optuna

- 최고의 예측 성능을 내기 위해 'Optuna' 사용
  - 성능 향상을 위해 각 ML 모델의 최적 하이퍼파라미터를 자동으로 탐색
  - AutoML의 한 영역으로 최근 각광

- 7개의 ML 모델 탐색
  - Random Forest, SVM, Neural Network, CNN, LightGBM, XGBoost, Stacking
  - 각 ML 모델마다 50번의 반복을 통해 다양한 하이퍼파라미터를 최적화

- 검증 데이터 분리
  - 성능 비교를 위한 검증 데이터 확보 (train_test_validation, Validation Size = 0.2)
  - 검증 세트를 통해 구한 Validation RMSE를 기준으로 사용

Random Forest and SVM

In [7]:
```
def objective_RF(trial):
    rf_1 = trial.suggest_int('max_depth', 2, 50)
    rf_2 = trial.suggest_int('min_samples_leaf', 1, 50)
    rf_3 = trial.suggest_int('n_estimators', 10, 1000)

    rf = RandomForestRegressor(max_depth=rf_1,
                               min_samples_leaf=rf_2,
                               n_estimators=rf_3)

    rf.fit(x_train, y_train)
    pred = rf.predict(x_val)

    rmse = mean_squared_error(pred, y_val, squared = False)

    with open("rf_{}.pickle".format(trial.number), "wb") as fout:
        pickle.dump(rf, fout)

    return rmse

seed_sampler = optuna.samplers.TPESampler(seed=seed)
study = optuna.create_study(direction = 'minimize', sampler=seed_sampler)
study.optimize(objective_RF, n_trials=50)
```

Random Forest

- **Best RMSE : 35.614093183308235**
- **Best Parameters**

: {'max_depth': 14, 'min_samples_leaf': 1, 'n_estimators': 414}

In [8]:
```
def objective_SVR(trial) :

    svm_2 = trial.suggest_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])
    svm_3 = trial.suggest_float('epsilon', 0.01, 5)
    svm_4 = trial.suggest_float('C', 0.3, 5)
    svm_5 = trial.suggest_int('degree', 1, 5)

    svm = SVR(kernel=svm_2, epsilon=svm_3, C=svm_4, degree=svm_5)
    svm.fit(x_train, y_train)
    pred = svm.predict(x_val)

    rmse = mean_squared_error(pred, y_val, squared = False)

    with open("svm_{}.pickle".format(trial.number), "wb") as fout:
        pickle.dump(svm, fout)

    return rmse

seed_sampler = optuna.samplers.TPESampler(seed=seed)
study = optuna.create_study(direction = 'minimize', sampler=seed_sampler)
study.optimize(objective_SVR, n_trials=50)
```

Support Vector Machine

- **Best RMSE : 49.643685111329674**
- **Best Parameters**

: {'kernel': 'linear', 'epsilon': 4.785593651456996, 'C': 0.3079903370090929, 'degree': 3}

Feedforward Neural Network

```
[12] def create_model(trial):
        n_hidden = trial.suggest_int('n_hidden', 1, 10) #hidden layers
        n_units = trial.suggest_int('n_units', 8, 128) #neuron per layer
        learning_rate = trial.suggest_float('learning_rate', 0.01, 0.2)

        model = Sequential()
        model.add(Dense(n_units, input_shape=(9, ), activation='relu')) #x_train.shape[1]
        model.add(BatchNormalization())
        for i in range(n_hidden):
            model.add(Dense(n_units, activation='relu'))
            model.add(Dropout(0.2))
        model.add(Dense(1))

        model.compile(loss= 'mean_squared_error',
                      optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate),
                      metrics=[metrics.mse])
        return model
```

NN 구조 선언

```
[19] def objective(trial):
        model = create_model(trial)

        epochs = trial.suggest_int('epoch',10, 100)
        batch_size = trial.suggest_int('batch_size',25, 75)

        with tf.device('/device:GPU:0'):
          model.fit(x_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size
                    ,verbose=0)

        y_pred = model.predict(x_val)
        rmse = mean_squared_error(y_val, y_pred, squared = False)
        print(rmse)
        return rmse
```

Optuna 적용

```
[20] study = optuna.create_study(direction = 'minimize')
     study.optimize(objective, n_trials=50, show_progress_bar=True)
```

- **Best RMSE: 42.7419254729894**
- **Best parameters: {'n_hidden': 1, 'n_units': 52, 'learning_rate': 0.11893801120593775, 'epoch': 47, 'batch_size': 39}**

Convolution Neural Network

In [12]:
```
def CNN1(trial):
    filters1 = trial.suggest_categorical('filters1',[128,256,300,400])
    filters2 = trial.suggest_categorical('filters2',[64,128,150,200])
    filters3 = trial.suggest_categorical('filters3',[32,64,75,100])
    Dropouts = trial.suggest_categorical('dropout',[0.1, 0.2, 0.3, 0.4, 0.5])
    optimizer = trial.suggest_categorical('optimizer',['Adam','SGD', 'Adagrad'])

    model = Sequential()
    model.add(Conv1D(filters=filters1, kernel_size = (1), activation='relu', input_shape = (1,9)))
    model.add(Dropout(Dropouts))
    model.add(MaxPooling1D(pool_size=(1)))

    model.add(Conv1D(filters2, kernel_size = (1), activation='relu'))
    model.add(Dropout(Dropouts))
    model.add(MaxPooling1D(pool_size=(1)))

    model.add(Conv1D(filters3, kernel_size = (1), activation='relu'))
    model.add(Dropout(Dropouts))
    model.add(MaxPooling1D(pool_size=(1)))
    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dense(32, activation = "relu"))
    model.add(Dense(16, activation = "relu"))
    model.add(Dense(1))

    model.compile(optimizer = optimizer, loss = "mean_squared_error", metrics=[metrics.mse])
    return model
```

CNN 구조 선언

In [13]:
```
def objective(trial):
    epochs = trial.suggest_int('epoch',10, 100)
    batch_size = trial.suggest_int('batch_size',25,75)
    rmsecv= list()

    model = CNN1(trial)

    t_x = x_train.reshape(x_train.shape[0],1, 9)
    t_y = np.ravel(y_train)
    val_x = x_val.reshape(x_val.shape[0],1,9)
    val_y = np.ravel(y_val)

    try:
        with tf.device('/device:GPU:0'):
            model.fit(t_x, t_y, epochs=epochs, batch_size=batch_size,
                      verbose=0, validation_data=(val_x, val_y))
        y_pred = model.predict(val_x)

        rmse = mean_squared_error(val_y, np.ravel(y_pred), squared=False)
        rmsecv.append(rmse)

    except:
        rmse = 100
    return rmse
```

Optuna 적용

- **Best RMSE: 39.44451260571045**
- **Best parameters: {'epoch': 85, 'batch_size': 30, 'filters1': 365, 'filters2': 128, 'filters3': 47, 'dropout': 0, 'optimizer': 'Adam'}**

LightGBM

```
def objective(trial):
    param = {
        'objective': 'regression',
        'metric': 'rmse',
        "verbosity": -1,
        "boosting_type": "gbdt",
#         "lambda_l1": trial.suggest_float("lambda_l1", 1e-4, 10.0, log=True),
#         "lambda_l2": trial.suggest_float("lambda_l2", 1e-4, 10.0, log=True),
        "num_leaves": trial.suggest_int("num_leaves", 2, 256),
        'max_depth': trial.suggest_int('max_depth',3, 50),
        'learning_rate': trial.suggest_float("learning_rate", 1e-4, 1.0, log=True),
        'min_child_samples': trial.suggest_int('min_child_samples', 5, 100),
        'subsample': trial.suggest_float('subsample', 0.4, 1, log=True)
    }
    # Train and Predict
    model = lgb.LGBMRegressor(**param)
    model.fit(x_train, y_train)
    pred = model.predict(x_val)

    pred = np.rint(pred)            # Return Integer
```

```
# Train and Predict
model = lgb.LGBMRegressor(**param)
model.fit(x_train, y_train)
pred = model.predict(x_val)

pred = np.rint(pred)            # Return Integer

# Accuracy
rmse = mean_squared_error(pred, y_val, squared = False)
nrmse = rmse / (y_val.max()-y_val.min())
# Save checkpoint
PATH = "/home/gynchoi/workspace/gayoon/ML/checkpoint/LightGBM/"
if not os.path.isdir(PATH):
    os.makedirs(PATH)
with open(PATH+"lgb_{}.pickle".format(trial.number), "wb") as fout:
    pickle.dump(model, fout)

return rmse
```

- **Best RMSE: 36.30973345795159**
- **Best parameters: {'num_leaves': 242, 'max_depth': 45, 'learning_rate': 0.07396441809284664, 'min_child_samples': 5, 'subsample': 0.5360325098726257}**

XGBoost

```python
def objective(trial):
    dtrain = xgb.DMatrix(x_train, label=y_train)
    dvalid = xgb.DMatrix(x_val, label=y_val)

    param = {
        "verbosity": 0,
        'objective': 'reg:linear',
        'eval_metric':'rmse',
        # use exact for small dataset.
        "tree_method": "exact",
        # defines booster, gblinear for linear functions.
        "booster": trial.suggest_categorical("booster", ["gbtree", "gblinear", "dart"]),
        # L2 regularization weight.
        "lambda": trial.suggest_float("lambda", 1e-4, 1.0, log=True),
        # L1 regularization weight.
        "alpha": trial.suggest_float("alpha", 1e-4, 1.0, log=True),
        # sampling ratio for training data.
        "subsample": trial.suggest_float("subsample", 0.2, 1.0),
        # sampling according to each tree.
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.2, 1.0),
    }

    if param["booster"] in ["gbtree", "dart"]:
        # maximum depth of the tree, signifies complexity of the tree.
        param["max_depth"] = trial.suggest_int("max_depth", 3, 30)#, step=2)
        # minimum child weight, larger the term more conservative the tree.
        param["min_child_weight"] = trial.suggest_int("min_child_weight", 0, 5)
        # learning rate
        param["eta"] = trial.suggest_float("eta", 1e-4, 1.0, log=True)
        # defines how selective algorithm is: min_split_loss
        param["gamma"] = trial.suggest_float("gamma", 1e-4, 1.0, log=True)
        param["grow_policy"] = trial.suggest_categorical("grow_policy", ["depthwise", "lossguide"])

    if param["booster"] == "dart":
        param["sample_type"] = trial.suggest_categorical("sample_type", ["uniform", "weighted"])
        param["normalize_type"] = trial.suggest_categorical("normalize_type", ["tree", "forest"])
        param["rate_drop"] = trial.suggest_float("rate_drop", 1e-4, 1.0, log=True)
        param["skip_drop"] = trial.suggest_float("skip_drop", 1e-4, 1.0, log=True)


    # Train and Predict
    model = xgb.XGBRegressor(**param)
    model.fit(x_train, y_train)

    pred = model.predict(x_val)
    pred = np.rint(pred)            # Return Integer
```

- **Best RMSE: 35.79491473890364**
- **Best parameters:**

```
Number of finished trials:  50
Best trial: 18
  Value: 35.79491473890364
  Params:
    booster: gbtree
    lambda: 0.0001024105051677 3672
    alpha: 0.004067534947839597
    subsample: 0.5666826636537622
    colsample_bytree: 0.8940848434037588
    max_depth: 16
    min_child_weight: 4
    eta: 0.04960838349694138
    gamma: 0.010895608289741043
    grow_policy: depthwise
```

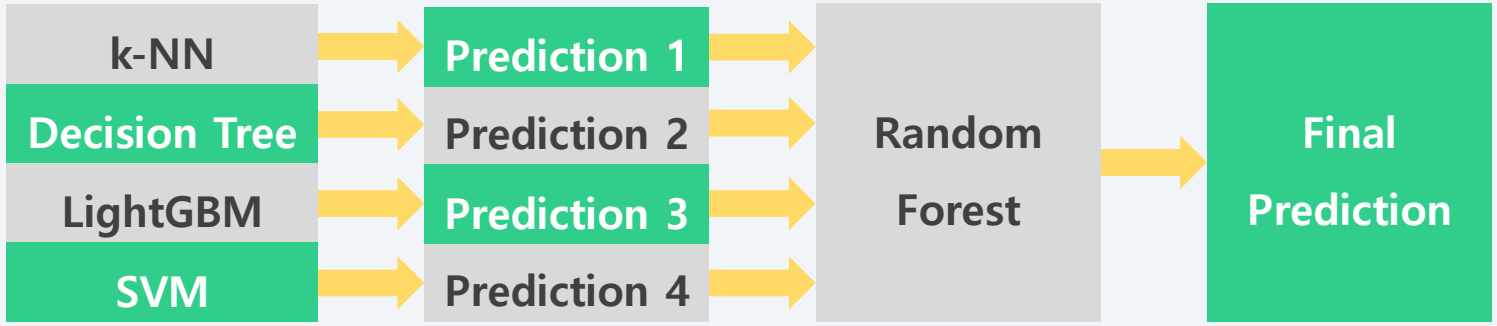
Stacking

```python
def objective(trial):

    knn_1 = trial.suggest_int('n_neighbors', 2, 20)
    knn_2 = trial.suggest_categorical('weights', ['uniform', 'distance'])
    knn = KNeighborsRegressor(n_neighbors=knn_1, weights=knn_2)

    dt_1 = trial.suggest_categorical('splitter', ['best', 'random'])
    dt_2 = trial.suggest_int('max_depth', 2, 50)
    dt_3 = trial.suggest_int('min_samples_leaf', 1, 50)
    dt = DecisionTreeRegressor(splitter=dt_1, max_depth=dt_2, min_samples_leaf=dt_3)

    dtrain = lgb.Dataset(x_train, label=y_train)
    dtest = lgb.Dataset(x_val)

    param = {
        'objective': 'regression',
        'metric': 'rmse',
        'max_depth': trial.suggest_int('max_depth',3, 15),
        'learning_rate': trial.suggest_float("learning_rate", 1e-8, 1e-2, log=True),
        'n_estimators': trial.suggest_int('n_estimators', 100, 3000),
        'min_child_samples': trial.suggest_int('min_child_samples', 5, 100),
        'subsample': trial.suggest_float('subsample', 0.4, 1, log=True)
    }
    gbm = lgb.LGBMRegressor(**param)
```

```python
svm_2 = trial.suggest_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])
svm_3 = trial.suggest_float('epsilon', 0.01, 1)
svm_4 = trial.suggest_float('C', 0.1, 2)
svm = SVR(kernel=svm_2, epsilon=svm_3, C=svm_4)

knn.fit(x_train, y_train)
dt.fit(x_train, y_train)
gbm.fit(x_train, y_train)
svm.fit(x_train, y_train)

knn_pred = knn.predict(x_val)
dt_pred = dt.predict(x_val)
gbm_pred = gbm.predict(x_val)
svm_pred = svm.predict(x_val)

pred = np.array([knn_pred, dt_pred, gbm_pred, svm_pred])
pred = np.transpose(pred)

rf_1 = trial.suggest_int('max_depth', 2, 50)
rf_2 = trial.suggest_int('min_samples_leaf', 1, 50)
rf_3 = trial.suggest_int('n_estimators', 10, 1000)
rf = RandomForestRegressor(max_depth=rf_1, min_samples_leaf=rf_2, n_estimators=rf_3)

rf.fit(pred, y_val)
final_pred = rf.predict(pred)
rmse = mean_squared_error(final_pred, y_val, squared = False)
```

- **Best RMSE: 14.747688355095349**
- **Best parameters: {'n_neighbors': 7, 'weights': 'distance', 'splitter': 'best', 'max_depth': 22, 'min_samples_leaf': 1, 'learning_rate': 0.0008557133802256295, 'n_estimators': 2988, 'min_child_samples': 59, 'subsample': 0.42705903884030194, 'kernel': 'rbf', 'epsilon': 0.9394191470253792, 'C': 1.8342301859640537}**

Stacking 구조:



Results

1

Stacking

RMSE: 14.747

Neural Network

RMSE: 42.741

CNN

RMSE: 39.444

LightGBM

RMSE: 36.309

XGBoost

RMSE: 35.794

SVR

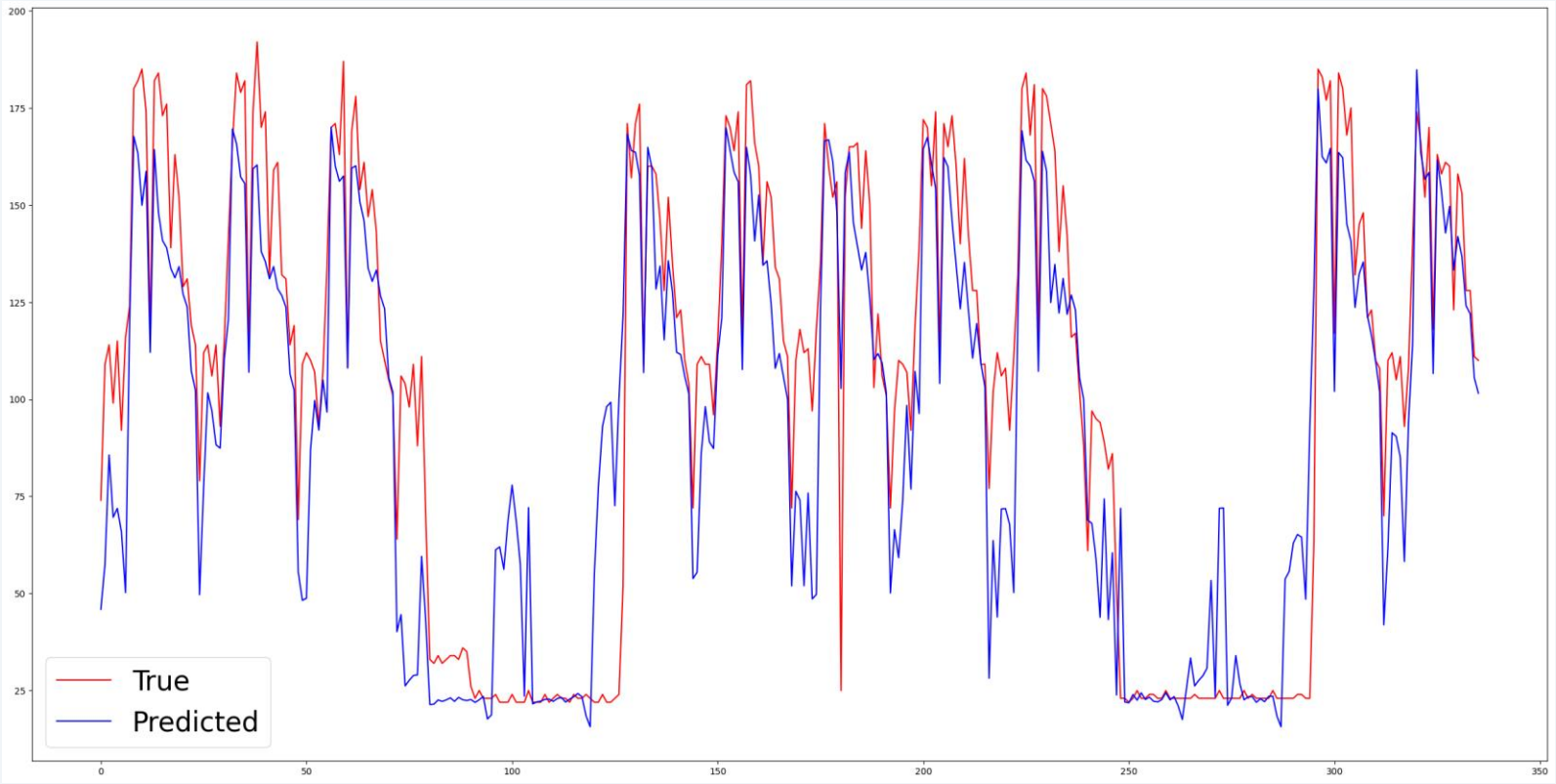
RMSE: 49.643

RF

RMSE: 35.614

- **결과 정리: 검증 RMSE는 Stacking에서 14.747로 가장 낮은 것을 확인할 수 있다.**
- **향후 최적화 모델에 학습된 Stacking 모델로 예측된 결과를 활용한다.**

Results



- 9월 데이터의 예측값과 실제값 시각화
- 예측값과 실제값은 RMSE 26 정도의 차이를 보였음

Optimization

Input 변수

변수의 목적 에 따라 분류 후 일부 선정

Prediction	시간 변수: 날짜, 시간, 15분, 30분, 45분, 60분, 평균, day, d, m 날씨 변수: 기온, 풍속, 습도, 강수량 전기요금 (계절)
Optimization	공장인원, 인건비, 생산량, 전기요금 **+ 예측된 시간당 평균 최대수요전력**

의사결정변수

- Index Set
  - t ∈ {1, 2, …, 336} (9/1 ~ 9/14)
- Decision Variable
  - y_t: t 시점에서 생산 여부, boolean
  - x_t: t 시점에서 생산 결정 시 생산량(기준: 10개), integer
    - x_t=1은 10개 생산 의미, x_t=2는 20개 생산 의미
    - 기준을 1개로 할 시, 대부분의 시점에서 1개만 생산하고 특정 경우에는 최대한으로 생산하는 문제 발생

매개변수

- Constant Parameters
  - m_1: 계획 기간 내 달성해야 하는 최소한의 총 생산량
  - m_2: 한 시점에서 생산할 수 있는 최대 생산량
  - d: 생산해야 하는 최소 시점의 수
  - c_e: 전기세 데이터 정규화 상수
  - c_h: 인건비 데이터 정규화 상수
  - α: 전기세의 비중(0~1 사이)
  - β: 인건비의 비중(0~1 사이)
    - α+β=1

매개변수

- Vectorized Parameters
  - $\hat{e}_t^d$: t 시점의 평균 최대전력수요 예측값
  - e_t^a: t 시점의 전기요금
  - h_t^r: t 시점의 공장인원
  - h_t^a: t 시점의 인건비
  - P_t^e: t 시점의 전기세, $\hat{e}_t^d * e_t^a / c_e$
  - P_t^h: t 시점의 인건비, $h_t^r * h_t^a / c_h$
  - P_t: t 시점의 총 비용, $\alpha * P_t^e + \beta * P_t^e$

최적화 모델

$$\min \sum_{t \in T} P_t \cdot x_t$$ → 목적 함수는 총 비용 합 최소화

$$\text{s.t.} \sum_{t \in T} x_t \geq m_1$$ → 9/1부터 9/14까지 최소 생산량을 달성해야 함

$$0 \leq x_t \leq m_2 \quad \forall t \in T$$ → 음의 생산은 불가,
한 번 생산 시 최대 생산량을 넘을 순 없음

$$x_t \geq y_t \quad \forall t \in T$$ → if $y_t = 0$ then $x_t = 0$

모델 구현 by CVXPY

```
edt = test_df['평균_예측'].values
eat = test_df['elec_cost'].values

hrt = test_df['인건비_변환'].values
hat = test_df['공장인원'].values

c1 = np.mean(edt*eat)
c2 = np.mean(hrt*hat)

pet = edt*eat/c1

pht = hrt*hat/c2

alpha = 0.5
beta = 0.5

pt = alpha*pet + beta*pht
```

평균 통해 데이터 정규화 => 두 비용의 단위를 같게
(인건비와 전기세 간 큰 차이가 없다는 가정)

인건비, 전기세 비중을 1:1로 설정

모델 구현 by CVXPY

```
x = cp.Variable(336, nonneg = True)
y = cp.Variable(336, boolean = True)

obj = cp.Minimize(sum([pt[i]*x[i] for i in range(336)]))

constraints = []

constraints += [sum([x[i] for i in range(336)]) >= 18000]
constraints += [sum([y[i] for i in range(336)]) >= 84]

for i in range(336):
    constraints += [x[i]<=600]
    constraints += [x[i]>=y[i]]

prob = cp.Problem(obj, constraints)

prob.solve()
```

```
2027.1820477911995
```

실제 데이터 기반으로
파라미터 값 설정

```
np.sum(df.iloc[-336:, 7]) #m_1
```

```
180764
```

```
np.max(df.iloc[-336:, 7]) #m_2
```

```
6214
```

```
336/4 #d
```

```
84.0
```

생산 계획 결과 확인

```
true_x = x.value*10
```

```
true_x
```

```
array([   0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0., 6000.,
       6000., 6000., 6000., 6000.,   10., 6000.,   10.,   10., 6000.,
         10., 6000.,   10.,   10., 6000., 6000.,   10.,   10.,   10.,
          0.,    0.,    0.,   10.,   10.,    0., 6000., 6000., 6000.,
         10.,   10., 6000.,   10.,   10., 6000.,   10.,   10.,   10.,
         10., 6000., 6000.,   10.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
         10.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,   10.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,   10.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
         10.,    0.,    0.,    0.,    0.,    0., 6000., 6000.,   10.,
       6000.,   10.,   10.,   10., 6000., 6000.,   10.,   10., 5460.,
         10., 6000., 6000.,   10.,   10.,   10.,   10.,   10.,   10.,
         10.,   10.,    0.,    0., 6000.,   10.,   10.,   10.,   10.,
         10.,   10., 6000.,   10., 6000.,   10.,   10., 6000., 6000.,
         10.,   10.,   10.,    0.,   10.,   10.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
          0.,    0.,    0.])
```

10단위로 변환 후 최적화된 생산 계획 확인

유의

- 최적화 모형 설계 단계에서 불확실성을 고려하지 않은 Deterministic LP를 사용하였고 따라서 현실에서 최적에 가까운 결과를 도출하기는 어려움