REPORT

과목: 기계학습
과제명: 회귀 예측 모델 구현 및 분석
학번: 202020933
이름: 송주훈
제출일: 2025-04-19

1. 프로젝트 개요

● 문제 정의

본 프로젝트의 목적은 건물의 다양한 구조적 특성과 설계 요소를 기반으로, **건물의 에너지 부하(난방 및 냉방 부하)를 예측하는 회귀 모델을 개발**하는 것이다.
두 가지 연속형 타겟 변수인 '난방 부하(Heating Load)'와 '냉방 부하(Cooling Load)'에 대해 각각 회귀 모델을 구축하고, 예측 성능을 비교·분석한다.

● 데이터셋 설명

- 출처: https://www.kaggle.com/datasets/ujjwalchowdhury/energy-efficiency-data-set/data
- 데이터셋 크기: 768개의 샘플 × 10개의 변수
- 종속변수 (Target):
    1. Heating_Load: 난방 부하 (연속형 실수 값)
    2. Cooling_Load: 냉방 부하 (연속형 실수 값)
- 주요 독립변수 (Features):
    Relative_Compactness, Surface_Area, Wall_Area, Roof_Area, Overall_Height, Orientation, Glazing_Area, Glazing_Area_Distribution

2. 데이터 전처리 및 탐색적 분석

● 결측치 처리

```
# 데이터 전처리를 위해 먼저 데이터에 결측치를 확인.
print("\n결측치 확인:")
print(df.isnull().sum())
```

```
결측치 확인:
Relative_Compactness        0
Surface_Area                0
Wall_Area                   0
Roof_Area                   0
Overall_Height              0
Orientation                 0
Glazing_Area                0
Glazing_Area_Distribution   0
Heating_Load                0
Cooling_Load                0
```

모든 변수에서 결측치는 존재하지 않음 (isnull().sum() 결과 전체 0).
→ 별도의 결측치 처리 없이 원본 데이터를 그대로 사용.

● 범주형 변수 처리

```
# 범주형 변수가 존재하는지 확인
categorical_cols = df.select_dtypes(include='object').columns
print("\n범주형 변수:", categorical_cols)
```

```
범주형 변수: Index([], dtype='object')
```

범주형 변수 인코딩을 진행하는 이유는 범주형 변수가 유한하고 고정된 수의 카테고리 또는

그룹을 나타내는 변수이기 때문이다.
따라서 범주형 데이터를 모델이 이해하고 처리할 수 있도록 수치 형태로 변환하는 과정이 필수적으로 진행되는데 이 과정이 바로 범주형 변수 인코딩 과정이다.
하지만 현재 확인 결과 데이터셋에 범주형 변수가 존재하지 않기 때문에 get_dummies 또는 OneHotEncoder를 진행하지 않고 넘어가도록 했다.

● 스케일링

먼저 스케일링의 필요 여부를 판단하기 위해 입력 변수의 요약 통계를 출력하도록 했다.

```
# 2. 스케일링 필요 여부 확인
print("입력 변수 요약 통계:")
display(X.describe())
```

	Relative_Compactness	Surface_Area	Wall_Area	Roof_Area	Overall_Height	Orientation	Glazing_Area	Glazing_Area_Distribution
count	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000	768.000000	768.00000
mean	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000	0.234375	2.81250
std	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763	0.133221	1.55096
min	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000	0.000000	0.00000
25%	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000	0.100000	1.75000
50%	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000	0.250000	3.00000
75%	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000	0.400000	4.00000
max	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000	0.400000	5.00000

확인 결과 변수 간 단위 차이가 매우 크다는 것을 확인했고 이는 학습에 영향을 줄 수 있기 때문에 StandardScaler를 사용하여 표준화(평균 0, 표준편차 1) 진행했다.

```
# 3. 스케일링(StandardScaler) 적용

# 스케일링을 위한 StandardScaler 클래스 가져오기
from sklearn.preprocessing import StandardScaler

# StandardScaler를 하나 만들기, 이는 평균은 0, 표준편차는 1로 값을 바꿔주는 도구임.
scaler = StandardScaler()

# 실제 X(입력 데이터)의 값을 스케일링 진행함.
# fit : 각 열의 평균과 표준편차를 계산하는 과정,  transform : 계산한 값을 기반으로 스케일링 적용
X_scaled = scaler.fit_transform(X)
```

그 결과 아래와 같이 mean은 거의 0에 가깝고, std는 거의 1에 가까우며, min, max는 -1 ~ 2 사이의 값을 갖는 것을 확인할 수 있다.

	Relative_Compactness	Surface_Area	Wall_Area	Roof_Area	Overall_Height	Orientation	Glazing_Area	Glazing_Area_Distribution
count	7.680000e+02	7.680000e+02	768.000000	7.680000e+02	768.000000	768.000000	7.680000e+02	768.000000
mean	-7.401487e-17	-4.163336e-16	0.000000	2.174187e-16	0.000000	0.000000	1.480297e-16	0.000000
std	1.000652e+00	1.000652e+00	1.000652	1.000652e+00	1.000652	1.000652	1.000652e+00	1.000652
min	-1.363812e+00	-1.785875e+00	-1.685854	-1.470077e+00	-1.000000	-1.341641	-1.760447e+00	-1.814575
25%	-7.725642e-01	-7.421818e-01	-0.561951	-7.915797e-01	-1.000000	-0.670820	-1.009323e+00	-0.685506
50%	-1.340162e-01	2.319318e-02	0.000000	1.583159e-01	0.000000	0.000000	1.173631e-01	0.120972
75%	6.227813e-01	7.885681e-01	0.561951	9.725122e-01	1.000000	0.670820	1.244049e+00	0.766154
max	2.041777e+00	1.553943e+00	2.247806	9.725122e-01	1.000000	1.341641	1.244049e+00	1.411336

● EDA 시각화 및 통계 요약

- 변수 간 상관관계 분석

```python
# 상관계수 행렬 계산
# df는 초기 데이터셋이고 df.corr()을 활용해 각 열(변수)들 사이의 상관관계를 계산하도록 함
corr_matrix = df.corr(numeric_only=True)

# 상관계수 출력
print("상관계수 행렬:")
print(corr_matrix)
```

위와 같이 먼저 df.corr()을 활용해 각 열들 사이에 상관관계를 계산하도록 진행하고 아래와 같이 히트맵을 활용하여 변수 간의 상관관계를 쉽게 확인할 수 있도록 진행했다.

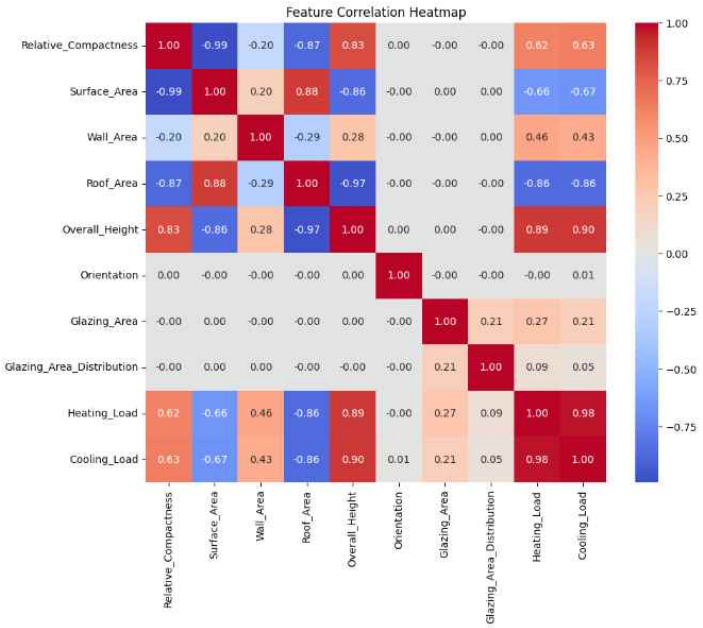
```python
import seaborn as sns
import matplotlib.pyplot as plt

# 위에서 계산한 상관관계를 토대로 히트맵을 활용하여 시각화를 진행

# 히트맵 크기 지정
plt.figure(figsize=(10, 8))

# sns.heatmap()을 활용하여 히트맵을 그림.
# corr_matrix : 상관관계 행렬,
# annot = True : 각 칸에 상관관계의 숫자를 같이 표시
# cmap='coolwarm' : 양의 상관관계 = 빨간색, 음의 상관관계 =  파란색 표시
# fmt=".2f" : 소수점 둘째 자리까지 표시시
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)


plt.title("Feature Correlation Heatmap")
plt.show()
```



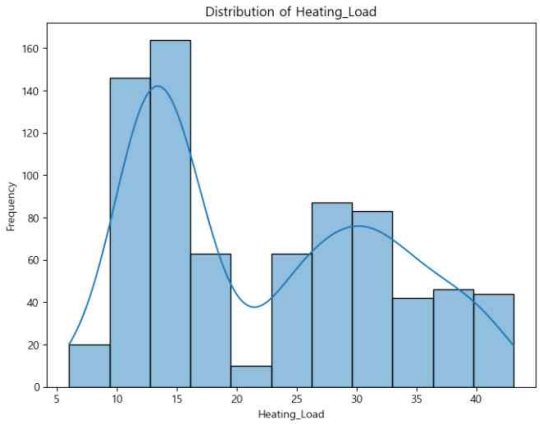
- 종속변수 분포 요약 (Heating Load / Cooling Load)

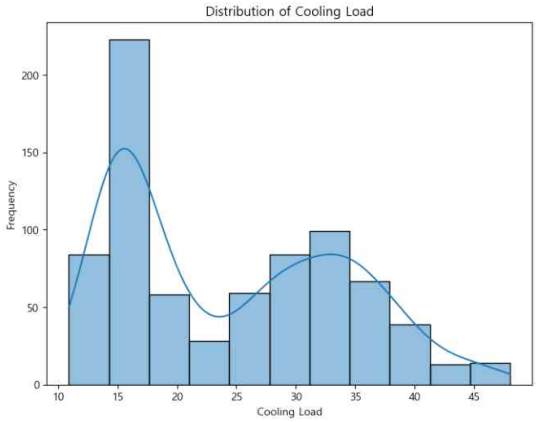
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

df_data = pd.DataFrame(df)

# Heating_Load 분포 시각화
plt.figure(figsize=(8, 6))
sns.histplot(df_data['Heating_Load'], kde=True)
plt.title('Distribution of Heating_Load')
plt.xlabel('Heating_Load')
plt.ylabel('Frequency')
plt.show()

# Cooling_Load 분포 시각화
plt.figure(figsize=(8, 6))
sns.histplot(df_data['Cooling_Load'], kde=True)
plt.title('Distribution of Cooling Load')
plt.xlabel('Cooling Load')
plt.ylabel('Frequency')
plt.show()
```





현재 데이터셋에는 2개의 종속 변수가 존재하기 때문에 다중 회귀 모델이 아닌 다음과 같이 2개의 회귀 모델을 만들기 위해 나눠서 시각화를 진행하도록 했다.

난방 부하(Heating_Load), 냉방 부하(Cooling_Load) => 두 타겟 모두 연속형이고 정규 분포에 가까운 형태를 띠며 이상치 존재하지 않는 것을 확인할 수 있다.

3. 모델 구축 및 학습

● 사용한 알고리즘

- 두 개의 종속변수(Heating_Load, Cooling_Load) 각각에 대해 2개의 회귀 모델을 구성하여 비교 분석을 진행한다:
    1. Linear Regression
    2. RandomForestRegressor
    → 각 모델은 개별적으로 두 타겟에 대해 학습 및 예측을 수행함
- 해당 모델을 선택한 이유는 현재 데이터가 변수 간 상관관계가 다양하고 복잡한 상호작용이 존재하기 때문이다. 따라서 가장 기본이 되는 모델인 Linear Regression과 여러 개의 결정 트리를 앙상블하여 평균을 도출하는 RandomForestRegressor 모델을 비교하고자 선택하였다.

● 데이터 분할 방식

- 모델의 일반화 성능을 공정하게 비교하기 위해 K-Fold 교차검증 (K=5) 사용하도록 했다.
- 각 fold에서 데이터가 랜덤하게 분할되어 학습/검증 과정 수행하도록 하였고 추가로 cross_val_score 및 cross_validate를 활용하여 정확한 성능을 측정하고자 했다.

● 파이프라인 사용 여부

- 각 모델별로 아래와 같은 Pipeline 구조를 사용하여 전처리와 모델 학습을 통합하도록 했다. 파이프라인 구조는 아래 코드와 같다.

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', model)
])
```

파이프라인의 구조가 위와 같은 이유는 다음과 같다.

- 이미 데이터 전처리 부분에서 스케일링을 진행했지만 이런 경우 X_test의 값이 X_train을 기준으로 정규화된 것이 아니라 X_test의 정보가 미리 모델에 들어가게 된다.
  즉 데이터 누수 발생하기 때문에 위와 같이 파이프라인을 구성하여 X_test가 자신을 기준으로 스케일링 되지 않고 X_train의 평균/표준편차만을 기준으로 변환되어 평가 데이터(X_test)의 정보가 모델 학습에 새어나가는 것을 막을 수 있도록 한 것이다.

● 학습 코드 요약

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, cross_validate
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler

# 공통 전처리 + 모델 파이프라인 정의
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LinearRegression())  # 또는 RandomForestRegressor()
```

```
])

# 교차검증 설정
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Heating_Load 또는 Cooling_Load에 대한 교차검증 평가
results = cross_validate(pipeline, X, y, cv=cv, scoring=('neg_root_mean_squared_error',
'neg_mean_absolute_error', 'r2'))
```

- 위 코드는 각 타겟에 대해 반복 실행되며, 모델별 성능을 비교할 수 있도록 구성되었다.

4. 성능 평가

● 사용한 지표

모델 성능 평가는 다음 세 가지 지표를 기반으로 수행됨:

- RMSE (Root Mean Squared Error): 평균 제곱 오차의 제곱근으로, 예측값과 실제값의 차이를 측정
- MAE (Mean Absolute Error): 예측값과 실제값의 절대 오차 평균
- R^2 Score (결정계수): 1에 가까울수록 높은 설명력을 가짐

● 모델별 성능 비교

□ Heating_Load 예측 성능

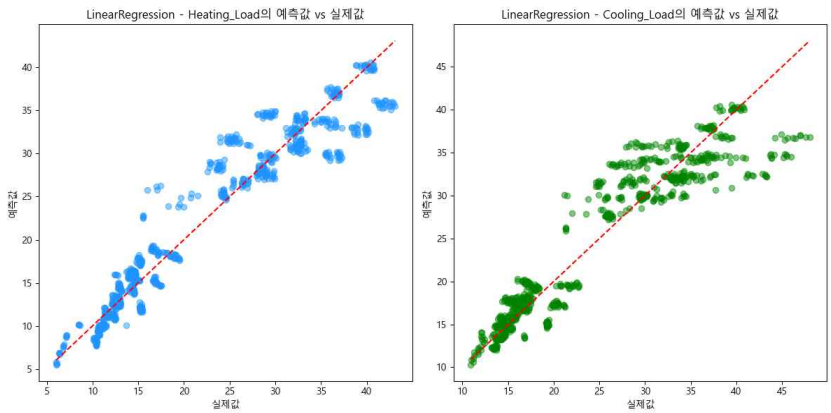
모델	RMSE	MAE	R^2
LinearRegression	2.9405	2.0828	0.9140
RandomForest	0.4756	0.3243	0.9977

□ Cooling_Load 예측 성능

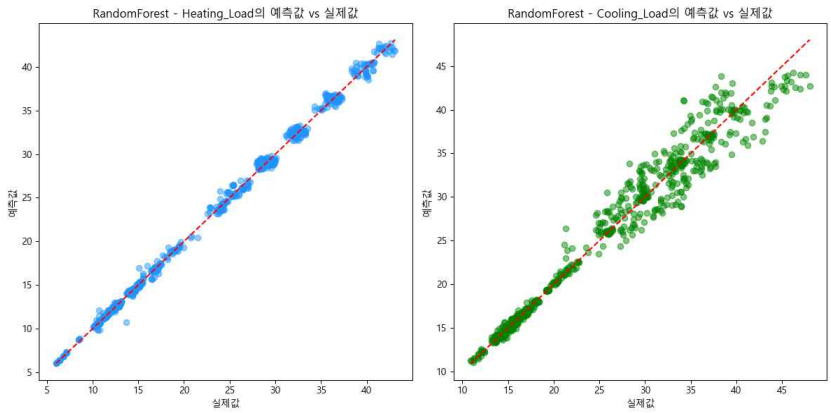
모델	RMSE	MAE	R^2
LinearRegression	3.2096	2.2649	0.8850
RandomForest	1.6327	1.0036	0.9700

종합적으로 볼 때, RandomForest 모델은 Heating_Load와 Cooling_Load 모두에 대해 LinearRegression 모델보다 훨씬 낮은 예측 오차를 보이며, 종속 변수의 변동성을 훨씬 더 잘 설명한다. 따라서 현재까지의 결과를 바탕으로는 RandomForest 모델이 더 잘 학습되었다는 것을 알 수 있다.

● 예측값 vs 실제값 시각화



Linear Regression은 전반적으로 잘 작동하지만, 일부 구간에서 과소/과대 예측 경향 존재한다.



Random Forest는 전반적으로 모든 구간에서 높은 정확도를 보이며, 예측 오차가 작다.

● 해석

- LinearRegression은 설명력은 높지만 RMSE/MAE가 비교적 크며, 복잡한 관계를 충분히 학습하지 못한다.
- RandomForestRegressor는 두 타겟 모두에서 매우 우수한 성능($R^2 > 0.96$) 을 기록하며, 비선형적 패턴을 효과적으로 학습한다.

5. 하이퍼파라미터 튜닝

● 튜닝 방법

- 사용 기법 : GridSearchCV
- 적용 모델 : LinearRegression, RandomForest

● 파라미터 그리드 및 파라미터 범위

```
lr_param_grid = {
    "lr__fit_intercept": [True, False],
    "lr__positive": [True, False]
}
```

LinearRegression에서 튜닝할 수 있는 소수의 파라미터로 다음과 같이 조정하도록 했다.

1. fit_intercept: 절편을 계산할 것인지 (보통 True가 적절)
2. positive: 회귀 계수를 양수로만 제한할지 여부 (특수한 경우에만 True)

```
rf_param_grid = {
    'rf__n_estimators': [100, 200, 300, 400, 500],
    'rf__max_depth': [None, 10, 20, 30, 40, 50],
    'rf__min_samples_split': [2, 5, 10, 15],    #
    'rf__min_samples_leaf': [1, 2, 4, 6]       # 
}
```

RandomForest의 주요 파라미터는 총 4개로 다음과 같이 조정하도록 했다.

1. n_estimators: 만들어질 트리의 개수 (많을수록 안정되지만 느려짐)
2. max_depth: 각 트리의 최대 깊이 (과적합 방지용)
3. min_samples_split: 노드를 분할하기 위한 최소 샘플 수 (값이 클수록 모델이 덜 복잡)
4. min_samples_leaf: 리프 노드에 있어야 할 최소 샘플 수 (값이 클수록 일반화 능력증가)

● 최적 하이퍼파라미터 및 성능

☐ Heating Load

LinearRegression

항목	값
최적 파라미터	'lr__fit_intercept': True, 'lr__positive': False
RMSE	2.9405
MAE	2.0828
R^2	0.9140

RandomForestRegressor

항목	값
최적 파라미터	'rf__max_depth': None, 'rf__min_samples_leaf': 1, 'rf__min_samples_split': 2, 'rf__n_estimators': 200
RMSE	0.4745
MAE	0.3240
R^2	0.9978

☐ Cooling Load

LinearRegression

항목	값
최적 파라미터	'lr__fit_intercept': True, 'lr__positive': False
RMSE	3.2096
MAE	2.2649
R^2	0.8850

RandomForestRegressor

항목	값
최적 파라미터	'rf__max_depth': None, 'rf__min_samples_leaf': 1, 'rf__min_samples_split': 2, 'rf__n_estimators': 500
RMSE	1.6262
MAE	1.0010
R^2	0.9703

● 튜닝 결과 분석

- LinearRegression 모델의 경우 이전과 비교하여 차이를 발견하지 못했다.
- 반면 RandomForest의 경우 GridSearchCV를 활용하여 4가지 파라미터인 n_estimators, max_depth, min_samples_leaf, min_samples_split에 대해 최적의 값을 찾아 이전과 비교하였을 때, RMSE와 MAE는 더 낮은 오차를 보였고 R^2의 경우 더 높은 설명력을 보여주었다.

6. 결론 및 고찰

● 최종 모델 성능 종합 평가

- 본 프로젝트에서는 Heating Load와 Cooling Load라는 두 개의 연속형 타겟을 예측하는 회귀 모델을 구축하였다.
- 여러 모델 중 RandomForestRegressor가 가장 높은 예측 정확도(R^2 > 0.96)와 낮은 오차를 보여주었다.
- 하이퍼파라미터 튜닝을 통해 각 타겟에 대해 최적의 설정을 적용함으로써 RMSE와 MAE가 추가로 감소하였고, R^2 > 0.97로 더 높은 정확도를 보여주었다.

● 데이터 또는 모델의 한계

- 데이터셋은 구조적으로 결측치가 없고 규모가 크지 않아 모델 학습에 적합했으나, 고차원적인 특성 간 상호작용을 고려하지는 못했다.
- 모델의 설명력이 높더라도 실제 에너지 효율에 영향을 미치는 외부 변수(예: 기후 조건 등)는 반영되지 못한다.

● 실생활 응용 가능성 또는 확장 방향

- 이 모델은 건축물의 구조적 특성을 바탕으로 건축 설계 단계에서 에너지 효율 예측에 활용 가능하다.
- 냉방/난방 부하를 정확히 예측함으로써 HVAC 시스템의 사전 설계 최적화에 도움을 줄 수 있다.
- 실제 응용을 위해서는 더 다양한 지역과 조건에서 수집된 현장 기반 데이터와의 통합이 필요하다.

● 다음 단계에서 고려할 점

- 특성 선택 기법을 통해 불필요한 변수 제거 및 성능 향상을 시도할 수 있다.
- 다양한 앙상블 기법을 활용하여 모델의 성능을 개선할 수 있다.

7. 참고자료

● 데이터셋 출처

https://www.kaggle.com/datasets/ujjwalchowdhury/energy-efficiency-data-set/data

● 참고 문헌

https://scikit-learn.org/stable/modules/preprocessing.html
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
https://velog.io/@jiazzang/%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%A0%84%EC%B2%98%EB%A6%AC-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%8A%A4%EC%BC%80%EC%9D%BC%EB%A7%81StandardScaler-MinMaxScaler-Robust
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics
https://wikidocs.net/145332
https://rudolf-2434.tistory.com/10

● 사용한 주요 라이브러리

- python 3.13.2
- pandas 2.2.3
- numpy 2.2.4
- matplotlib 3.10.1
- seaborn 0.13.2
- scikit-learn 1.6.1

● 중요 코드 스니펫

- 파이프 라인 정의 코드

```
# 파이프라인 정의
lr_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LinearRegression())
])
```

```
rf_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("rf", RandomForestRegressor(random_state=42))
])
```

- 모델 평가 루프 코드

```
# cross_val_predict 사용하여 예측값 추출
y_pred_Heating_Load = cross_val_predict(pipeline, X, y_Heating_Load, cv=kf)

y_pred_Cooling_Load = cross_val_predict(pipeline, X, y_Cooling_Load, cv=kf)
```

```
# Heating_Load에 대한 평가 지표 계산
rmse_scores_H = cross_val_score(pipeline, X, y_Heating_Load, cv=kf, scoring=rmse_scorer)
mae_scores_H = cross_val_score(pipeline, X, y_Heating_Load, cv=kf, scoring=mae_scorer)
r2_scores_H = cross_val_score(pipeline, X, y_Heating_Load, cv=kf, scoring=r2_scorer)

# Cooling_Load에 대한 평가 지표 계산
rmse_scores_C = cross_val_score(pipeline, X, y_Cooling_Load, cv=kf, scoring=rmse_scorer)
mae_scores_C = cross_val_score(pipeline, X, y_Cooling_Load, cv=kf, scoring=mae_scorer)
r2_scores_C = cross_val_score(pipeline, X, y_Cooling_Load, cv=kf, scoring=r2_scorer)
```

- GridSearchCV 설정 코드

```python
lr_param_grid = {
    "lr__fit_intercept": [True, False],
    "lr__positive": [True, False]
}

# GridSearchCV로 튜닝
lr_grid = GridSearchCV(lr_pipeline, lr_param_grid, cv=kf, scoring=rmse_scorer)
lr_grid.fit(X, target_data)
```

```python
rf_param_grid = {
    'rf__n_estimators': [100, 200, 300, 400, 500],     # 트리 수 (모델 안정성 증가 vs 속도 감소)
    'rf__max_depth': [None, 10, 20, 30, 40, 50],        # 트리 최대 깊이 (None이면 제한 없음 → 과적합 위험 증가)
    'rf__min_samples_split': [2, 5, 10, 15],    # 내부 노드 분할 최소 샘플 수
    'rf__min_samples_leaf': [1, 2, 4, 6]       # 리프 노드 최소 샘플 수 (값이 클수록 과적합 방지)
}

for target_name, target_data in zip(["Heating_Load", "Cooling_Load"], [y_Heating_Load, y_Cooling_Load]):

    rf_pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("rf", RandomForestRegressor(random_state=42))
    ])

    rf_grid = GridSearchCV(rf_pipeline, rf_param_grid, cv=kf, scoring=rmse_scorer, n_jobs=-1)
    rf_grid.fit(X, target_data)
```