기상청

날씨마루 분석환경을 활용한 빅데이터 분석 교육실습자료

전력기상지수 분석환경 사용법

기상청

Contents

분석 개요

기상청

본 분석 실습은 한국전력공사(KEPCO) 에서 제공하는 전력·기상 융합 데이터를 사용하여,
날씨와 시간 등 다양한 요인이 공동주택의 전력 사용량에 미치는 영향을 알아보고,
이를 기반으로 미래의 전력 수요를 예측하는 모델을 구축합니다.

전력·기상 융합 데이터

- 한국 전력공사와 기상청이 공동주택 부하관리 연구개발 활성화를 목적으로 제공하는 데이터셋입니다.
- 시간대별 공동주택 전력 부하와 기상 관측값을 포함하며, 안정적인 전력 수급 예측의 기초자료로 활용됩니다.

전력 수요 예측

- 과거의 전력 사용량과 관련 변수(기온, 요일, 시간 등)의 패턴을 학습하여 미래의 전력 사용량을 예측하는 시계열 분석 문제입니다.
- 정확한 예측은 전력망의 안정적 운영과 에너지 효율화에 필수적입니다.

분석 시나리오

기상청

분석 과정은 입력 데이터의 이해부터 시작하여, 변수 생성, 모델 구축, 예측 성능 평가에 이르기까지 단계적으로 구성되어 있습니다. 특히 시간, 기온, 습도 등 다양한 요인이 전력 소비에 어떤 영향을 미치는지 시각적으로 탐색하고, 이를 바탕으로 예측모델을 설계합니다.

입력 데이터

01 기상 데이터: 기온, 습도, 풍속 등

02 전력 데이터: 과거 전력 사용량, 계약 전력 등

03 시간 정보: 연도, 월, 시, 요일 등

1단계: 전력 수요 영향 변수 선택

- 정의: 전력 사용량에 영향을 미치는 주요 변수들의 패턴을 탐색하고, 예측에 유용한 새로운 변수(특성)을 생성
- 분석 방법: 시각적 데이터 분석(EDA), 특성 공학

2단계: 전력 수요 예측 모형 구축

- 정의: 머신러닝 기법(Random Forest)을 활용하여 시간, 날짜 등 조건에 따른 전력 사용량을 예측하는 모형을 구축
- 분석방법: Random Forest Regressor
- 분석결과: 미래 시점의 전력 사용량 예측치

분석 절차

기상청

분석 절차는 총 다섯 단계로 구성되어 있으며, 데이터 로딩부터 시작해 데이터 처리, 탐색적 분석, 모델 학습 및 검증, 결과 해석 순으로 진행됩니다. 각 단계는 실제 전력 수요 예측 모델을 개발하는 데 필요한 전 과정을 체계적으로 포함하고 있습니다. 특히 각 과정에서의 핵심 작업과 주요 도구들이 무엇인지 간결하게 정리되어 있습니다.

데이터 로딩

분석 환경을 설정하고 전력 및 기상 데이터 준비

- 분석 환경 설정 및 패키지 로딩
- 데이터 불러오기 및 샘플링

데이터 처리

데이터를 분석에 적합하게 가공하고, 성능 향상을 위한 변수 생성

- 시계열 데이터 변환, 결측치 처리, 특성 공학

데이터 탐색

정제된 데이터의 분포 및 패턴을 시각적으로 확인

- 데이터 분포 시각화
- 데이터 패턴 분석

모형 구축

데이터를 분할하고 여러 모델을 학습시켜 성능 비교

- 학습/검증용 데이터 분할 및 스케일링
- 4개 머신러닝 모델 학습 및 평가

모형 검증

최종 모델을 상세히 분석하고 예측 결과를 시각화

- 변수 중요도 분석
- 예측 결과 시각화 및 성능 평가

분석 데이터 변수 정보

분석에 활용된 데이터는 시간 정보, 전력 정보, 기상 정보로 구분되며,
각 변수의 이름과 의미를 표 형태로 정리하였습니다. 예측 대상은 '전력부하합계'이며,
과거 전력 소비, 계약 전력, 기온, 습도, 풍속 등의 다양한 입력 변수가 이를 설명하는 데 사용됩니다.

입력 변수 목록 및 형식

- 변수명(column)"은 데이터셋 내 열 순서를 기준으로 표기한 것
- 실제 CSV 파일에는 변수명이 명시되어 있지 않으며, 열 번호를 통해 각 변수의 위치를 나타냄

카테고리	변수	설명	형식
위치 정보	X격자, Y격자	기상청 동네예보 X축, Y축 격자번호	Int
	위도, 경도	격자 위도, 경도 좌표	Float
시간 정보	연도, 월, 일, 시	측정 연도, 월, 일, 시각(01~24시)	Int
전력 정보	계약전력합계	격자내 공동주택 계약전력 합계(단위: kW)	Int
	공동주택수	격자내 공동주택 수(단위: 단지)	Int
	전력부하합계 (예측 대상)	격자내 해당시각 전력부하 합계(단위: kWh)	Float
기상 정보	기온	기온 관측값 (단위: °C)	Float
	상대습도	상대습도 관측값 (단위: %)	Float
	풍속	풍속 관측값(단위: m/s)	Float

기상청

Ⅰ. 데이터 로딩

1. 분석 환경 설정 및 패키지 로딩
2. 데이터 불러오기
3. 데이터 샘플링
4. 데이터 구조확인

1. 분석 환경 설정 및 패키지 로딩

기상청

분석에 사용되는 주요 데이터 변수들의 상세 정보를 제공하며, 이들은 시간, 전력, 기상 정보로 분류됩니다. 또한, 데이터 분석을 위한 필수 Python 패키지들을 주피터 노트북 환경에 로딩하는 초기 환경 설정 과정을 설명합니다. `pandas`, `matplotlib`, `seaborn`, `numpy` 등 핵심 라이브러리들은 데이터 처리, 시각화, 수치 연산에 필수적인 도구들입니다.

패키지 설치 및 로딩

- 분석에 사용할 패키지를 주피터 노트북 환경에서 로딩

```
# --- 라이브러리 임포트 및 환경 설정 ---
import os
import pandas as pd     # 데이터 분석을 위한 라이브러리
import numpy as np      # 다차원 배열을 위한 라이브러리
import matplotlib.pyplot as plt # 시각화를 위한 라이브러리
import seaborn as sns   # 시각화를 위한 라이브러리
import koreanize_matplotlib # 한글 폰트 깨짐 방지
import warnings

# 모델링 및 평가 관련 라이브러리
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_percentage_error
```

2. 데이터 불러오기

기상청

파일 경로를 지정하여 전력 데이터를 불러오고, info() 함수를 통해 총 행의 수를 확인하고
데이터의 구조와 함께 '기온', '상대습도', '풍속' 컬럼에서 결측치가 있음을 확인합니다.
이 과정은 한국전력공사(KEPCO)에서 제공하는 전력·기상 융합 데이터를 활용하여
미래 전력 수요를 예측하는 모델을 구축하기 위한 필수적인 첫 단계입니다.

데이터를 불러온 뒤, 구조 확인하기

- 준비된 데이터를 판다스의 read_csv함수를 이용해 불러온 후, .info() 함수로 구조 확인

```
# --- 데이터 로딩 ---
print("₩n--- 데이터 로딩 ---")
DATA_FOLDER = 'data'
csv_files = [f for f in os.listdir(DATA_FOLDER) if f.endswith('.csv')]
if not csv_files:
    raise FileNotFoundError(f"'{DATA_FOLDER}' 폴더에 분석할 CSV 파일이 없습니다.")

df_list = []
for csv_file in csv_files:
    file_path = os.path.join(DATA_FOLDER, csv_file)
    try:
        df_temp = pd.read_csv(file_path, encoding="cp949")
        df_list.append(df_temp)
    except Exception as e:
        print(f"'{csv_file}' 파일 로딩 실패: {e}")
```

3. 데이터 샘플링

기상청

해당 코드는 전체 원본 데이터에서 일부 데이터를 무작위로 추출하여 새로운 데이터프레임을 생성하는 과정입니다. 분석 대상 데이터가 수십만 건에 달하기 때문에, 연산 효율성과 메모리 사용량을 고려하여 일정 비율만 샘플링하였습니다.

데이터 샘플링

- 원활한 분석을 위한 10% 데이터 샘플링

```
# 이미 10% 샘플링된 600k 데이터 사용
df_raw = pd.concat(df_list, ignore_index=True)
df_raw.to_csv("data/kepco_ml_600k.csv", index=False)
print(f"데이터 로딩 및 통합 저장 완료. 총 {len(df_raw)}개 행.")
```

실행 결과

- 600,674개의 데이터 로드 완료

```
--- 데이터 로딩 —

데이터 로딩 완료. 총 600674개 행.
```

4. 데이터 구조확인

기상청

샘플링된 데이터에 대해 info() 함수를 통해 각 컬럼의 결측치 여부와 데이터 유형을 다시 확인합니다.
이를 통해 전처리의 필요성과 우선순위를 결정할 수 있으며,
특히 분석에서 중요한 기온, 습도, 풍속 등의 컬럼에 결측치가 포함되어 있는 점을 확인합니다.

데이터 구조

- 원활한 분석을 위해 데이터의 전체적인 형태 및 메모리 사용량 확인

```
df_raw.info()
```

실행 결과

- 기온, 상대습도, 풍속 컬럼에서 non-null 개수가 전체 행의 개수보다 적은 것을 통해 결측치 (누락된 값)가 존재함을 확인

```
Data columns (total 14 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   X격자     600674 non-null  int64
 1   Y격자     600674 non-null  int64
 2   위도      600674 non-null  float64
 3   경도      600674 non-null  float64
 4   연도      600674 non-null  int64
 5   월       600674 non-null  int64
 6   일       600674 non-null  int64
 7   시       600674 non-null  int64
 8   계약전력합계  600674 non-null  int64
 9   공동주택수   600674 non-null  int64
 10  전력부하합계  600674 non-null  float64
 11  기온      561949 non-null  float64
 12  상대습도    509639 non-null  float64
 13  풍속      560977 non-null  float64
dtypes: float64(6), int64(8)
memory usage: 64.2 MB
```

기상청

Ⅱ. 데이터 처리

1. 시계열 데이터 변환
2. 결측치 처리
3. 특성 공학 (Feature Engineering)

1. 시계열 데이터 변환

원본 데이터에는 연도, 월, 일, 시 등의 컬럼이 개별적으로 존재하지만, 시계열 분석을 위해 이들을 하나의 datetime 형태로 통합합니다. 시간의 흐름이나 주기성을 반영하려면 단순한 숫자가 아닌 시간 개념으로 처리하는 것이 필수입니다. 변환된 datetime은 분석의 기준 인덱스로 설정되어 이후 작업의 기반이 됩니다.

시계열 데이터 변환

- 시계열 데이터 변환의 필요성

  현재 '연', '월', '일', '시' 컬럼은 단순한 숫자에 불과하여, 컴퓨터가 시간의 연속성과 주기성을 이해하지 못함

  시계열 분석을 위해, 이를 표준 datetime 형식으로 통합

```
# '시' 컬럼이 24시인 경우, 0시로 변경
df_raw['시'] = df_raw['시'].replace(24, 0)

# --- datetime 객체 생성 및 인덱스 설정 ---
# 각 컬럼을 문자열로 변환 후, 지정된 format으로 datetime 객체 생성
df_raw['datetime'] = pd.to_datetime(
    df_raw['연도'].astype(str) + '-' +
    df_raw['월'].astype(str).str.zfill(2) + '-' +
    df_raw['일'].astype(str).str.zfill(2) + ' ' +
    df_raw['시'].astype(str).str.zfill(2) + ':00:00',
    format='%Y-%m-%d %H:%M:%S'
)
# 생성된 datetime을 데이터프레임의 인덱스로 설정
df_processed = df_raw.set_index('datetime')
```

2. 결측치 처리

기상청

기온, 습도, 풍속 등의 기상 변수에는 일부 결측치가 존재하며, 시계열 데이터 특성을 고려해 바로 이전 시간의 값으로 채우는 ffill(Forward Fill) 방식으로 처리합니다. 이는 기상 요소가 급격히 변하지 않는다는 전제 하에, 가장 합리적인 추정치를 제공하는 전략입니다.
결측치가 제거되어 분석의 신뢰도를 높이는 기반이 마련됩니다.

시계열 데이터에서의 결측치 처리 전략

- 방법: Forward Fill (ffill) - 누락된 값을 바로 이전시간대의 값으로 채움
- 이유: 기온, 습도와 같은 기상데이터는 급격히 변하지 않으므로, 바로 이전 시간의 데이터가 가장 합리적인 추정치

```
# 결측치 확인 및 처리 (시계열 데이터이므로 ffill 사용)
print("결측치 처리 전:₩n", df_processed.isnull().sum().loc[lambda x: x > 0])
df_processed.fillna(method='ffill', inplace=True)
print("₩n결측치 처리 완료.")
```

실행 결과

- 결측치 처리 완료

```
결측치 처리 전:
기온   38725
상대습도   91035
풍속   39697
dtype:  int64

결측치 처리 완료.
```

3. 특성 공학 - 주기성 변환

기상청

'월'과 '시'와 같은 변수는 순환적인 성격을 가지지만, 숫자로 표현되면 12월과 1월이 멀리 떨어진 값으로 인식됩니다. 이를 보완하기 위해 Sin/Cos 변환을 통해 주기성을 수학적으로 표현하여 모델이 시간적 패턴을 인식할 수 있도록 합니다. 이러한 변환은 예측 성능 향상에 유의미한 효과를 줍니다.

주기성 특성 (Sin/Cos 변환)

- 문제점: 모델은 12월과 1월이 가깝다는 사실을 모름 (숫자 12와 1은 멀다고 인식)
- 해결책: '월'과 '시' 데이터를 삼각함수(Sin/Cos)로 변환하여, 시간의 순환적 특성을 모델이 학습하도록 함

```
# 주기성 특성 (Sin/Cos 변환)
df_processed['시간_sin'] = np.sin(2 * np.pi * df_processed['시'] / 24)
df_processed['시간_cos'] = np.cos(2 * np.pi * df_processed['시'] / 24)
df_processed['월_sin'] = np.sin(2 * np.pi * df_processed['월'] / 12)
df_processed['월_cos'] = np.cos(2 * np.pi * df_processed['월'] / 12)
```

3. 특성 공학 - 파생 변수

기상청

전력 사용량에 영향을 줄 수 있는 요인을 반영하기 위해 다양한 도메인 지식을 기반으로 새로운 변수를 생성합니다. 예를 들어 '주말 여부', '냉방도일', '불쾌지수' 등은 생활 패턴과 기후 요소가 전력 수요에 미치는 영향을 구체적으로 반영한 변수입니다.
이는 모델이 환경과 행동 특성을 더 잘 이해하도록 돕습니다.

분석가의 지식을 활용한 파생 변수 생성

- 주말: 주중과 주말의 뚜렷한 생활 패턴 차이를 반영
- 냉방도일 / 난방도일: 특정 기준 온도를 넘어설 때의 값을 계산하여, 냉/난방 수요를 명시적으로 표현
- 불쾌지수: 기온과 습도를 조합하여 여름철 전력 수요와 관련 높은 변수를 생성, 톰(Thom)의 불쾌지수 공식을 섭씨(℃) 온도 기준으로 변형

$$\text{불쾌지수} = 1.8 \times \text{기온} - 0.55 \times (1 - \text{상대습도} / 100) \times (1.8 \times \text{기온} - 26) + 32$$

```
# 파생 변수
df_processed['주말'] = df_processed['요일'].apply(lambda x: 1 if x >= 5 else 0)

df_processed['냉방도일'] = np.maximum(0, df_processed['기온'] - 24)

df_processed['난방도일'] = np.maximum(0, df_processed['기온'] - 18)

df_processed['불쾌지수'] = 1.8 * df_processed['기온']
- 0.55 * (1 - df_processed['상대습도']/100) * (1.8 * df_processed['기온'] - 26) + 32
```

3. 특성 공학 - 시계열 특성

기상청

과거 데이터는 미래를 예측하는 데 있어 매우 중요한 단서가 됩니다.
이를 반영하기 위해 전력 수요와 기온 등의 lag(지연) 특성과, 최근 시간 동안의 평균값을 반영하는 rolling(이동 평균) 특성이 도입됩니다. 예를 들어 하루 전, 일주일 전 전력 사용량이나 최근 24시간 기온 평균은 매우 강력한 예측 변수로 활용됩니다.

과거의 데이터에서 힌트 얻기

- Lag Feature: “하루 전, 일주일 전 같은 시간의 전력 사용량은 어땠을까?”
  과거의 데이터는 미래 예측의 강력한 단서
  1일(24시간), 1주일(168시간) 전의데이터를 가져옴

- Rolling Feature: “최근 24시간 동안의 평균 기온은 어땠을까?”
  데이터이 단기적인 추세를 반영하기 위해 이동평균(Rolling Mean) 변수를 추가

```
# 시계열 특성 (Lag, Rolling)
for lag in [24, 168]: # 1일, 1주일 전 데이터
    df_processed[f'전력부하_lag_{lag}h'] = df_processed[TARGET_COLUMN].shift(lag)
    df_processed[f'기온_lag_{lag}h'] = df_processed['기온'].shift(lag)
df_processed['기온_rolling_24h'] = df_processed['기온'].rolling(window=24).mean()
```

3. 특성 공학 - 최종 확인

기상청

Lag와 Rolling 특성을 생성하면 데이터 앞부분에 결측치가 생기는데,
이를 bfill(backward fill) 방식으로 처리하여 모든 행의 누락값을 채웁니다. 최종적으로 특성 수가 기존의 14개에서 28개로 확장되며, 모델이 학습할 수 있는 정보의 폭이 크게 넓어집니다.

특성 공학 완료

- Lag, Rolling 특성 생성 시 데이터의 맨 앞부분에는 계산할 과거 값이 없어 결측치가 발생
- 이는 이전 값으로 채우는 bfill (backward fill) 방식으로 처리

```
# 특성 생성으로 인한 결측치는 이전 값으로 채움
df_processed.fillna(method='bfill', inplace=True)
df_featured = df_processed.copy()
print("특성 공학 완료. 최종 특성 개수:", len(df_featured.columns))
```

실행 결과

- 최종적으로 모델이 학습에 사용할 수 있는 변수가 기존 14개에서 28개로 확장

```
특성 공학 완료. 최종 특성 개수: 28
```

기상청

III. 데이터 탐색 (EDA)

1. 데이터 분포 시각화
2. 시계열 패턴 분석
3. 상관관계 분석

1. 데이터 분포 시각화

기상청

데이터에 포함된 주요 변수들의 전체적인 분포를 히스토그램으로 시각화하여 확인합니다.
이러한 분포 형태를 사전에 파악하는 것은 이상값(outlier) 존재 여부를 감지하고, 변수 변환 여부, 적절한 모델 선택에 대한 판단 기준을 제공합니다.

주요 변수들을 히스토그램으로 시각화

- 공간 정보: X격자, Y격자, 위도, 경도
  X, Y격자와 위도, 경도 그래프를 보면, 데이터가 대한민국 전역에 고르게 분포하는것이 아니라, 특정 지역에 집중되어 있음을 알 수 있음

- 시간 정보: 연도, 월, 일, 시
  연도는 3개년에 걸쳐 비교적 고르게, 월, 일, 시는 거의 완벽하게 균등한 분포를 보임

- 전력 설비 및 사용량 정보: 계약전력합계, 공동주택수, 전력부하합계
  모두 오른쪽으로 긴 꼬리를 가진(Right-skewed) 분포를 보임

- 기상 정보: 기온, 상대습도, 풍속
  각각 독특한 분포를 보임

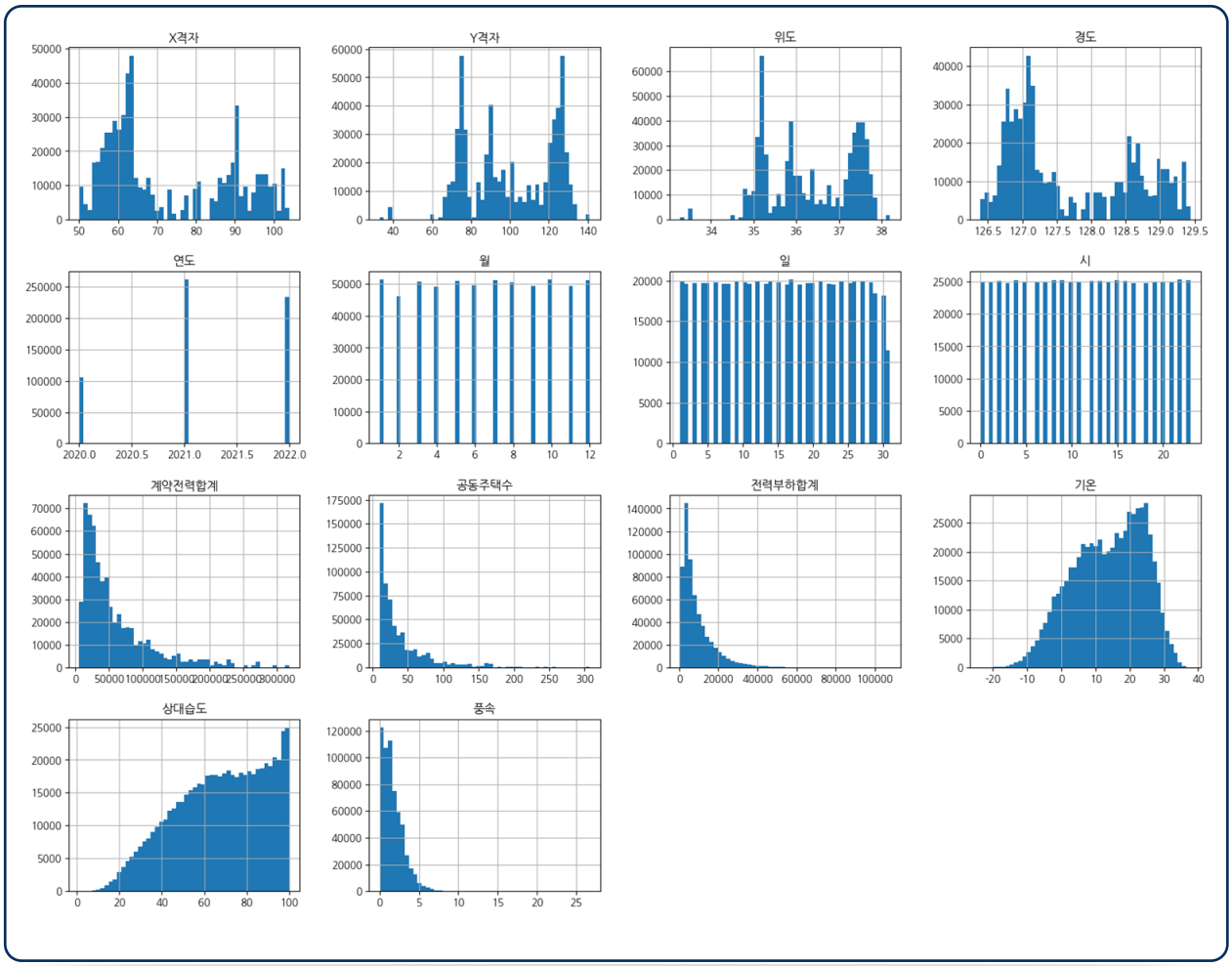
시각화 코드

```
#변수별 분포 히스토그램
df_processed.hist(bins=50, figsize=(20, 15))

#변수에 대한 요약 통계표
df_processed.describe()
```

1. 데이터 분포 시각화

실행 결과

◆ 수치형 변수별 분포를 나타낸 히스토그램 시각화



1. 데이터 분포 시각화

기상청

실행 결과

◆ 수치형 변수에 대한 요약 통계표

	X격자	Y격자	위도	경도	연도	월	일	시	계약전력합계	공동주택수	전력부하합계	기온	상대습도	풍속
count	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000	600674.000000
mean	72.705929	100.294491	36.337769	127.688261	2021.214591	6.527611	15.736316	11.506956	58702.626556	39.255666	8880.187428	13.718135	67.530748	1.700581
std	16.036248	22.178332	1.028793	0.896784	0.719766	3.450324	8.799870	6.923309	52444.383212	38.798147	8124.775443	10.353259	21.444910	1.365113
min	50.000000	33.000000	33.273132	126.390373	2020.000000	1.000000	1.000000	0.000000	5475.000000	10.000000	372.040000	-23.700000	1.200000	0.000000
25%	59.000000	77.000000	35.251380	126.931838	2021.000000	4.000000	8.000000	6.000000	22800.000000	15.000000	3375.850000	5.900000	51.600000	0.700000
50%	65.000000	97.000000	36.197530	127.254762	2021.000000	7.000000	16.000000	12.000000	40000.000000	25.000000	6070.360000	14.700000	69.300000	1.400000
75%	89.000000	123.000000	37.395995	128.586234	2022.000000	10.000000	23.000000	18.000000	76735.000000	48.000000	11741.670000	22.300000	86.000000	2.400000
max	104.000000	141.000000	38.198058	129.438071	2022.000000	12.000000	31.000000	23.000000	318552.000000	307.000000	107780.080000	37.700000	100.000000	26.700000

기술 통계표 요약 및 해석

◆ 데이터의 규모 일관성

count(개수): 모든 변수의 count가 600,674개로 동일

이는 데이터에 결측치가 하나도 없으며, 데이터가 잘 정제되었음을 보여줌

◆ 주요 변수별 특징

기온: 최저(min): -23.7℃, 최고(max): 37.7℃로 대한민국의 겨울과 여름철 기온 특성을 잘 반영하고 있음

전력부하합계: 평균 약 8,880이지만 최댓값은 107,780으로 매우큼

또한 평균 8,880이 중앙값 6,070보다 큼

상대습도: 최저 1.2%, 최고 100%로 매우 건조한 날부터 비가 오는 날까지 모든 상황의 데이터가 포함되어 있음

2. 시계열 패턴 분석

기상청

월별 및 시간대별 평균 전력 사용량을 시각화 함으로써 전력 수요의 시계열적 특성을 분석합니다. 이러한 분석은 시간에 따른 전력 사용의 변화 양상을 정량적으로 이해하고, 향후 시계열 모델이 시간 변수를 적절히 활용할 수 있도록 설계 방향을 제시하는 역할을 합니다.

월별 및 시간별 평균 전력 사용량

```
# 시각화를 통해 데이터 패턴 파악 (분석 대상 연도 기준)
df_eda = df_processed.copy()
fig, axes = plt.subplots(1, 2, figsize=(18, 6))
# 월별 평균 전력부하
df_eda.groupby(
    df_eda.index.month
    )['전력부하합계'].mean().plot(ax=axes[0], marker='o', title='월별 평균 전력부하합계')
axes[0].set_xlabel('월')
axes[0].set_ylabel('평균 전력부하합계')
axes[0].grid(True)
# 시간대별 평균 전력부하
df_eda.groupby(
    df_eda.index.hour
    )['전력부하합계'].mean().plot(ax=axes[1], marker='o', title='시간대별 평균 전력부하합계')
axes[1].set_xlabel('시')
axes[1].set_ylabel('평균 전력부하합계')
axes[1].grid(True)
plt.tight_layout()
plt.show()
print("시간에 따른 전력부하 패턴 시각화")
```

2. 시계열 패턴 분석

기상청

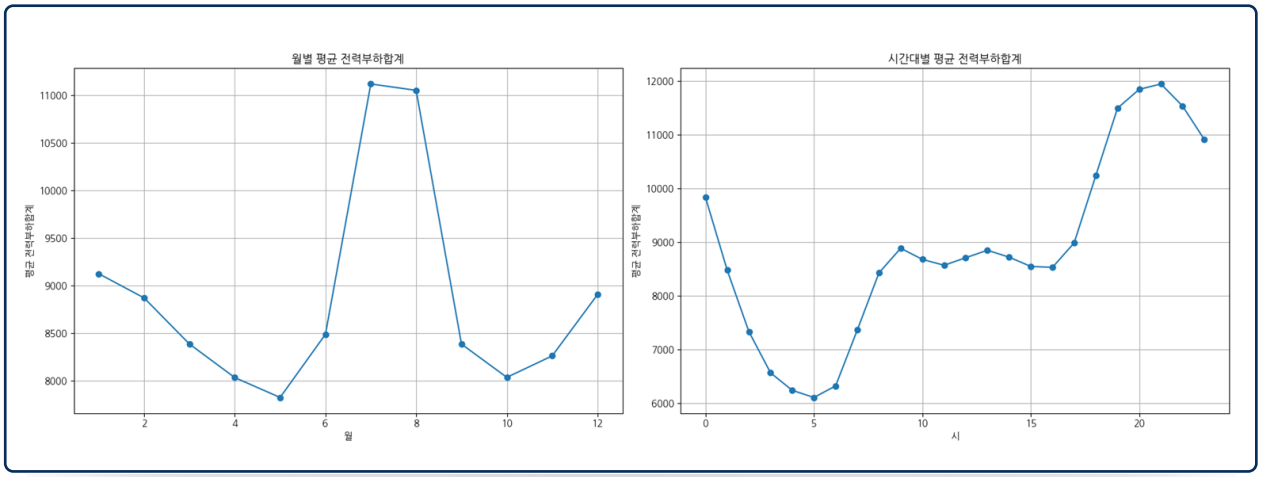
전력 사용량의 시간적 추세를 확인하기 위해 월별 및 시간대별 평균 전력 수요를 시각화하였습니다.
선형 그래프를 사용하여, 시간에 따른 전력 소비의 흐름과 경향성을 파악할 수 있고,
이러한 경향은 시계열 기반 예측 모델에서 시간 변수의 중요성을 뒷받침하며,
주기성 특성을 반영한 특성 공학 설계의 근거로 활용될 수 있습니다

월별 패턴 : 계절성

- 전력 사용량은 여름철(7-8)월과 겨울철(12-1월)에 정점을 찍는 U자 형태의 뚜렷한 계절성을 보임
- 이는 냉방과 난방 수요가 전력 사용의 핵심 요인임을 의미

시간별 패턴 : 주기성

- 하루 중 전력 사용량은 오전(7-9시)과 저녁(19-21시)에 두 개의 봉우리를 가짐
- 이는 출근 준비와 퇴근 후 저녁 활동이라는 공동주택의 전형적인 생활 패턴을 반영



3. 상관관계 분석

두 변수 간의 선형적 관계를 측정하는 통계 기법인 '피어슨 상관계수'를 사용합니다. 이 분석을 통해 어떤 변수가 예측 목표인 '전력부하합계'와 얼마나 밀접한 관련이 있는지 정량적으로 파악하고, 예측 모델의 핵심이 될 변수들을 식별합니다.

상관관계 분석이란?

- 두 변수 X와 Y가 함께 변화하는 경향을 -1부터 1사이의 값으로 나타내는 분석 방법
- 값의 의미:

  +1: 완벽한 양의 선형 관계 (X가 증가하면 Y도 비례하여 증가)

  -1: 완벽한 음의 선형 관계 (X가 증가하면 Y는 비례하여 감소)

  0: 선형 관계 없음

```
#상관관계 분석
print("₩n--- 상관관계 분석 ---")

# 상관관계 행렬 계산 (숫자형 데이터만 대상으로)
corr_matrix = df_processed.corr(numeric_only=True)

# 1. 히트맵 시각화
print("₩n[상관관계 히트맵]")
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, cmap='coolwarm')
plt.title('상관관계 히트맵', fontsize=16)
plt.show()

# 2. '전력부하합계'와의 상관계수 확인
print("₩n['전력부하합계'와의 상관계수]")
corr_target = corr_matrix['전력부하합계'].sort_values(ascending=False)
print(corr_target)
```

3. 상관관계 분석

상관관계 분석 결과

- 단순 상관계수 분석 결과, '계약전력합계'와 '공동주택수'는 목표 변수와 매우 높은 양의 상관관계를 보임
- 하지만 이는 독립변수 간 강한 상관관계, 즉 다중공선성(Multicollinearity) 문제를 간과한 해석일 수 있으며, 이 경우 각 변수의 실제 중요도가 왜곡될 수 있어 주의가 필요

```
['전력부하합계'와의 상관계수]
전력부하합계    1.000000
계약전력합계    0.905547
공동주택수     0.832367
위도         0.253206
Y격자        0.252688
시          0.162751
기온         0.068057
연도         0.012758
월          0.005561
일         -0.001693
풍속        -0.039022
상대습도     -0.043023
경도        -0.171699
X격자       -0.172971
Name: 전력부하합계, dtype: float64
```

기상청

Ⅳ. 모형 구축

1. 분석 데이터 셋 분할
2. 특성과 타겟 분리 및 스케일링
3. 비교 모델 학습 및 평가
4. 교차 검증

1. 분석 데이터 셋 분할

기상청

시계열 데이터의 특성상 학습용과 평가용 데이터를 무작위로 분리하는 것은 적절하지 않으며, 시간 순서를 고려한 분할이 필수적입니다. 이에 따라 전체 데이터 중 앞부분 80%를 훈련세트로, 나머지 20%를 테스트 세트로 설정하였으며, 이러한 방식은 실제 예측 상황과 유사하게 과거 데이터를 기반으로 미래를 예측하는 구조를 유지함으로써, 데이터 누수(Data Leakage)를 방지하고 모델 성능을 객관적으로 평가할 수 있는 환경을 제공합니다.

분석 데이터 셋 분할 - 개념 및 코드

- 시계열 데이터 분할의 원칙 : 과거로 미래를 예측

  Why? 시간의 흐름에 따른 패턴을 학습해야 하므로, 미래의 정보로 과거를 예측하는 상황을 만들면 안됨 (Data Leakage 방지)

  How? 따라서 데이터를 무작위로 섞지 않고, 특정 시점을 기준으로 과거(Train set)와 미래(Test set)로 명확히 분리

- 분할 전략 : 80% (학습용) vs 20% (평가용)

  전체 기간의 앞 80% 데이터는 모델을 학습시키는데 사용하고, 뒤 20% 데이터는 학습된 모델이 얼마나 잘 예측하는지 성능을 검증하는데 사용

```
# --- 시계열 데이터를 훈련/테스트 세트로 분할 ---
# 전체 데이터의 80% 지점의 인덱스를 찾습니다.
split_index = int(len(df_featured) * 0.8)
# 해당 인덱스의 날짜를 분할 기준으로 설정합니다.
SPLIT_DATE = df_featured.index[split_index]

# 분할 기준일 이전 데이터는 훈련 세트로 사용합니다.
train = df_featured.loc[df_featured.index < SPLIT_DATE]
# 분할 기준일 이후 데이터는 테스트 세트로 사용합니다.
test = df_featured.loc[df_featured.index >= SPLIT_DATE]

# 분할된 데이터의 기간과 크기를 출력하여 확인합니다.
print(f"훈련 데이터: {train.index.min()} ~ {train.index.max()} ({len(train)}개 행)")
print(f"테스트 데이터: {test.index.min()} ~ {test.index.max()} ({len(test)}개 행)")
```

2. 특성과 타겟 분리 및 스케일링

기상청

예측 모델 학습을 위해 입력 변수(X)와 목표 변수(y)를 분리하였습니다. 입력 변수는 전력 수요에 영향을 줄 수 있는 다양한 특성들을 포함하며, 목표 변수는 전력부하합계로 설정됩니다.
또한 모델 학습의 안정성 확보와 특정 변수의 영향력이 과도하게 작용하는 것을 방지하기 위해, 모든 입력 변수에 대해 표준화(Standard Scaling)를 적용하였습니다.

1단계: 특성(x)과 타겟(y) 분리

- X(Features): 예측에 사용할 모든 입력 변수 (27개)
- y(Target): 예측하고자 하는 정답 값 (전력부하합계)

2단계: 데이터 스케일링 (표준화)

- Why? 계약전력합계 (수십만 단위)와 기온 (십 단위) 처럼 변수마다 값의 범위가 크게 다르면, 모델이 큰 값을 가진 변수에 더 많은 가중치를 부여하는 등 학습이 왜곡될 수 있음
- What? StandardScaler를 사용하여 모든 특성(X) 변수들의 평균을 0, 표준편차를 1로 만들어, 모든 변수가 동등한 조건에서 모델에 영향을 주도록 함
- [매우 중요] 훈련 세트(X_train)에서 계산된 평균과 표준편차를 테스트 세트(X_test)에도 그대로 적용해야 함

```
# --- 특성(X)과 타겟(y) 분리 ---
features = [col for col in df_featured.columns if col != TARGET_COLUMN]
X_train, y_train = train[features], train[TARGET_COLUMN]
X_test, y_test = test[features], test[TARGET_COLUMN]

# --- 데이터 스케일링 ---
scaler = StandardScaler()
# 훈련 데이터로 스케일러를 학습(fit)시키고 변환(transform)합니다.
X_train_scaled = scaler.fit_transform(X_train)
# 학습된 스케일러로 테스트 데이터를 변환(transform)만 합니다.
X_test_scaled = scaler.transform(X_test)
```

3. 비교 모델 학습 및 평가

기상청

예측 정확도를 높이기 위해 서로 다른 방식의 머신러닝 모델 네 가지를 비교 학습하였습니다.
여러 모델을 학습하고 동일한 조건에서 평가함으로써, 실제 환경에 적합한 최적의 예측 모델을 객관적으로 선정할 수 있습니다.

1. Linear Regression (선형 회귀)

- 가장 기본적인 '직선' 관계를 찾는 모델
- 데이터의 기본적인 추세를 파악하는데 용이

2. Ridge (릿지 회귀)

- 선형 회귀에 '과적합 방지' 기능을 추가하여 모델의 안정성을 높임

3. Random Forest (랜덤 포레스트)

- 수많은 '의사결정 나무'를 만들어 투표(평균)하는 방식
- 개별 나무의 단점을 보완하고 예측력을 극대화

4. Gradient Boosting (그래디언트 부스팅)

- 모델이 순차적으로 이전 모델의 실수를 보완하며 학습하는 '엘리트 학습' 방식
- 일반적으로 매우 높은 성능을 보임

3. 비교 모델 학습 및 평가

기상청

각 모델은 훈련 데이터로 학습된 후 테스트 데이터에 대한 예측값을 생성하고,
이에 대해 3가지 지표로 성능을 정량 평가하였습니다.
이와 같은 평가 방식은 다양한 알고리즘의 상대적 강점을 비교하고,
데이터 특성에 가장 적합한 모델을 식별하는데 유용합니다.

반복문을 이용한 4개 모델 동시 학습 및 평가

- for 반복문을 사용하여 4개의 모델을 순서대로 불러와 학습(fit)과 예측(predict), 그리고 평가(metrics)를 자동으로 수행

```
# 비교할 모델들을 딕셔너리 형태로 정의합니다.
models = {
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42,
n_jobs=-1),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42)
}

results_list = [] # 평가 결과를 저장할 빈 리스트 생성
# 딕셔너리의 모델을 하나씩 꺼내어 반복합니다.
for name, model in models.items():
    # 1. 모델 학습: 스케일링된 훈련 데이터로 모델을 학습시킵니다.
    model.fit(X_train_scaled, y_train)
    # 2. 예측: 학습된 모델로 테스트 데이터의 y값을 예측합니다.
    y_pred = model.predict(X_test_scaled)
    # 3. 평가: 실제값(y_test)과 예측값(y_pred)을 비교하여 성능 지표를 계산합니다.
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mape = mean_absolute_percentage_error(y_test, y_pred) * 100
    # 4. 결과 저장: 계산된 성능 지표를 리스트에 추가합니다.
    results_list.append({'Model': name, 'R2': r2, 'RMSE': rmse, 'MAPE(%)': mape})

# 저장된 결과 리스트를 보기 좋은 데이터프레임으로 변환합니다.
results_df = pd.DataFrame(results_list).set_index('Model')
```

3. 비교 모델 학습 및 평가

기상청

모든 모델에 대해 동일한 기준으로 평가를 수행한 결과, Random Forest 모델이 가장 높은 설명력과 낮은 오차율을 기록하였습니다. 반면, 선형 회귀 계열 모델은 다소 낮은 성능을 보였으며,
이는 데이터의 비선형적 특성과 다중 변수간 상호작용이 복합적으로 작용한 결과로 해석됩니다.

평가지표

- R^2 (설명력): 모델이 얼마나 데이터의 변화를 잘 설명하였는가? (1에 가까울수록 높은 성능)
- RMSE (오차 크기): 예측이 평균적으로 얼마나 틀렸는가? (0에 가까울수록 높은 성능)
- MAPE (오차율): 평균 오차를 백분율로 환산한 값. (0에 가까울수록 높은 성능)

실행 결과

Model	R^2	RMSE	MAPE
Random Forest	0.965710	1392.824073	7.629671
Gradient Boosting	0.947662	1720.756181	15.442730
Linear Regression	0.862323	2790.884991	30.322476
Ridge	0.862319	2790.922717	30.297387

4. 교차 검증

앞서 보여드린 8:2 분할 방식은 특정 기간(마지막 20%)의 데이터에만 모델 성능이 좌우될 수 있는 한계가 있습니다. 우연히 그 기간이 쉽거나 어려웠다면, 모델의 성능을 과대/과소 평가할 수 있습니다. 따라서 시계열 교차 검증을 도입하여, 모델의 일반화 성능과 안정성을 객관적으로 검증합니다.

시계열 교차 검증 (Time Series Cross-Validation)

- 데이터를 시간 순서에 따라 여러 겹(Fold)로 나눈 후, 점점 더 많은 과거 데이터로 학습하고, 바로 다음 미래 구간을 예측하는 과정을 반복합니다.
- 이 과정을 통해 얻은 여러 개의 평가 점수의 평균을 최종 성능으로 삼아, 훨씬 더 안정적이고 신뢰도 높은 평가가 가능합니다.

```
from sklearn.model_selection import TimeSeriesSplit, cross_validate
from sklearn.metrics import make_scorer, r2_score, mean_squared_error,
mean_absolute_percentage_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import numpy as np

# 특성과 타겟 정의
X = df_processed.drop(columns=['전력부하합계'])
y = df_processed['전력부하합계']

# 스케일링
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# RMSE 함수 정의
def rmse_score(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# 평가 지표 정의
scoring = {
    'R2': make_scorer(r2_score),
    'RMSE': make_scorer(rmse_score),
    'MAPE': make_scorer(mean_absolute_percentage_error)
}
```

4. 교차 검증

기상청

```
# 시계열 교차검증 설정
tscv = TimeSeriesSplit(n_splits=5)

# 기존 모델
models_cv = {
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42,
n_jobs=-1),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42)
}

# 교차검증 실행
print("₩n--- 시계열 교차검증 결과 ---")
for name, model in models_cv.items():
    print(f"==== {name} ====")
    results = cross_validate(model, X_scaled, y, cv=tscv, scoring=scoring)
    for metric in scoring.keys():
        scores = results[f'test_{metric}']
        print(f"{metric}: mean = {scores.mean():.4f}, std = {scores.std():.4f}, scores = {scores}")
    print()
```

실행 결과

- 교차 검증 특성상 초반은 학습데이터 부족 등을 이유로 전반적인 평균 성능이 떨어지는 모습을 보이지만, 모델별 성능 순위는 유지되어 일관된 경향이 관찰됨
  이는 해당 모델의 일반화 성능이 데이터셋 분할에 크게 의존하지 않음을 시사

Model	R^2	RMSE	MAPE
Random Forest	0.9441	1925.6306	10.78
Gradient Boosting	0.9351	2105.2491	15.19
Linear Regression	0.8731	2946.5507	29.8
Ridge	0.8736	2940.9298	29.52

4. 교차 검증

시계열 교차 검증 상세 결과

- TimeSeriesSplit을 적용하여 각 모델의 일반화 성능을 5-Fold 기준으로 평가
- 모든 모델에 대해 R^2, RMSE, MAPE 지표를 평균 및 표준편차와 함께 제시

--- 시계열 교차검증 결과 ---

==== Linear Regression ====

R2: mean = 0.8731, std = 0.0212, scores = [0.89378608 0.83873598 0.89674588 0.86447986 0.87200186]

RMSE: mean = 2946.5507, std = 632.1415, scores = [2802.2551562 4198.54888135 2595.37084672 2558.218213 2578.36037693]

MAPE: mean = 0.2980, std = 0.0132, scores = [0.28876011 0.30156448 0.28616804 0.32213166 0.29130114]

==== Ridge ====

R2: mean = 0.8736, std = 0.0216, scores = [0.89592026 0.83873462 0.89671577 0.86451338 0.87199076]

RMSE: mean = 2940.9298, std = 633.5243, scores = [2773.95916743 4198.56659483 2595.74924903 2557.90179953 2578.47223963]

MAPE: mean = 0.2952, std = 0.0153, scores = [0.276942 0.30071132 0.28581026 0.32161301 0.29107963]

==== Random Forest ====

R2: mean = 0.9441, std = 0.0343, scores = [0.90376537 0.90194411 0.9756587 0.95968066 0.97951981]

RMSE: mean = 1925.6306, std = 882.2489, scores = [2667.36627158 3273.91434815 1260.13653666 1395.37954535 1031.35635131]

MAPE: mean = 0.1078, std = 0.0429, scores = [0.18928778 0.10043974 0.08025572 0.10274675 0.06640067]

==== Gradient Boosting ====

R2: mean = 0.9351, std = 0.0247, scores = [0.90706524 0.90618343 0.96509678 0.93979478 0.95733636]

RMSE: mean = 2105.2491, std = 687.8736, scores = [2621.23558948 3202.36071594 1508.96254082 1705.11136054 1488.5750627]

MAPE: mean = 0.1519, std = 0.0120, scores = [0.17465322 0.15156346 0.14157057 0.14929212 0.1423195]

기상청

V. 모형 구축

1. 최종 모형 심층 분석
2. 예측 성능 시각화
3. 결론
4. 향후 과제

1. 최종 모형 심층 분석

기상청

최종 예측 모델로 선정된 Random Forest Regressor는 트리 기반 앙상블 기법으로, 각 특성이 예측 결과에 기여한 정도를 정량적으로 산출할 수 있습니다. 변수 중요도는 전체 예측 과정에서 해당 변수가 얼마나 자주, 얼마나 효과적으로 분기 기준으로 사용되었는지를 기반으로 계산됩니다.

최종 모델 선정 : Random Forest Regressor

- 모든 평가지표 (R^2, RMSE, MAPE)에서 가장 우수한 성능을 기록

변수 중요도(Feature Importance) 분석

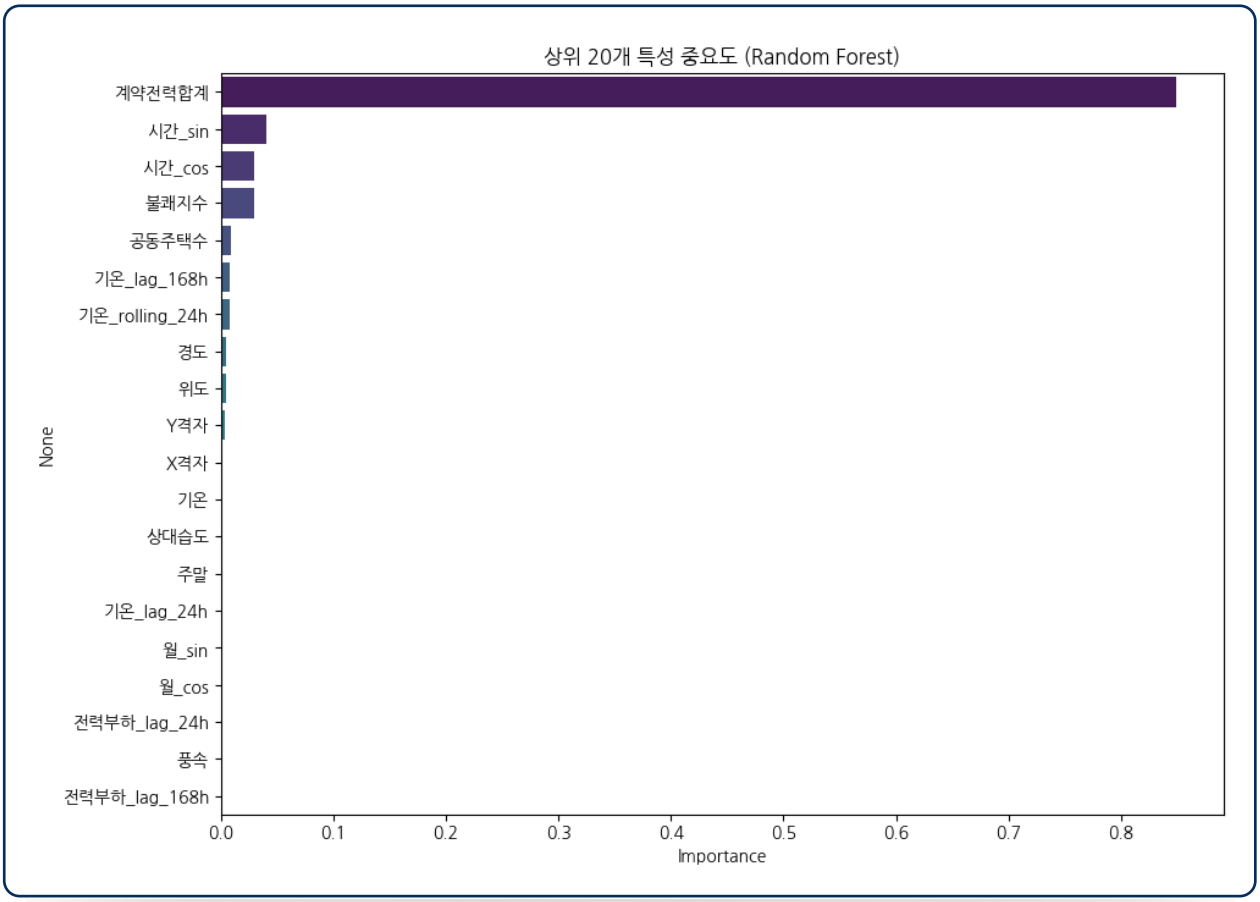
- 변수 중요도: 최종 선정된 Random Forest 모델이 예측값을 만들 때, 수많은 변수들 중 어떤 변수를 더 비중있게 고려했는지를 수치화한 지표
- 모델의 예측 과정을 이해하고, 어떤 변수가 실제 현상(전력 수요)에 큰 영향을 미치는지에 대한 통찰을 얻을 수 있음

```
# --- 최종 모델 선정 및 변수 중요도 분석 ---
# 최종 모델을 다시 정의하고 전체 훈련 데이터로 학습합니다.
final_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
final_model.fit(X_train_scaled, y_train)

# 변수 중요도를 추출하여 데이터프레임으로 만듭니다.
importances = pd.Series(final_model.feature_importances_, index=features)
top_20_importances = importances.sort_values(ascending=False).head(20)

# 변수 중요도를 시각화합니다.
plt.figure(figsize=(10, 8))
sns.barplot(x=top_20_importances.values, y=top_20_importances.index, palette='viridis')
plt.title('상위 20개 특성 중요도 (Random Forest)', fontsize=16)
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```

1. 최종 모형 심층 분석

상위 20개의 주요 변수를 시각화 한 결과, 특성 공학을 통해 새롭게 생성한 변수들이 높은 중요도를 차지한 것으로 확인되었습니다. 이는 원본 변수뿐 아니라 파생 변수들이 모델의 성능 향상에 실질적인 기여를 하고 있음을 의미합니다.

실행 결과



분석 결과

- 가장 중요한 변수 : 계약전력합계
- 특성 공학의 효과 : 변수 중요도 상위권에 시간_sin/cos(시간주기성), 기온_rolling_24h(최근 기온 추세), 불쾌지수 등 직접 만든 특성 공학 변수들이 대거 포진

2. 예측 성능 시각화 - 전체 기간

기상청

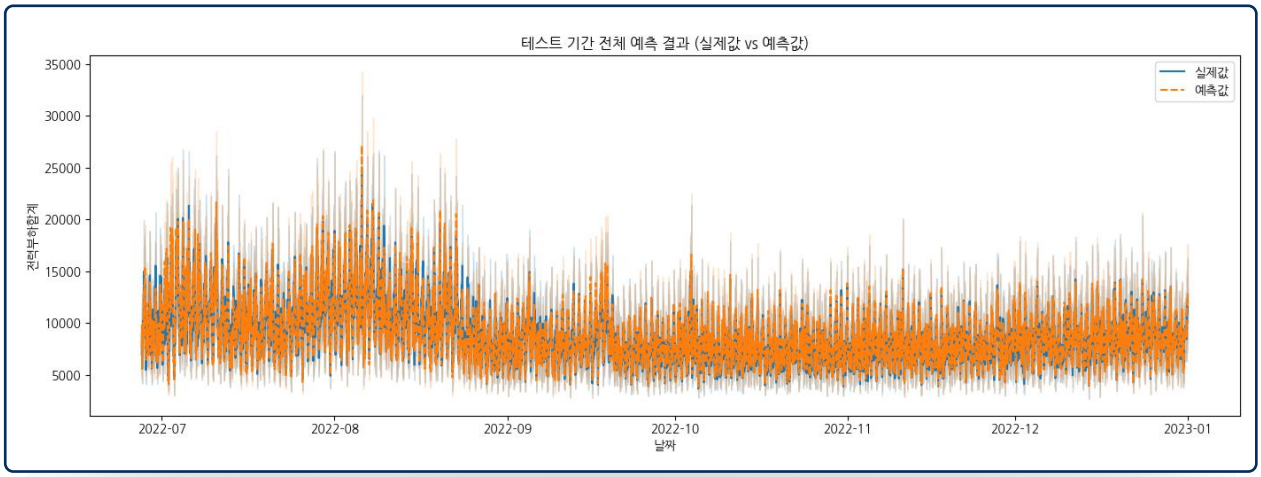
최종 모델의 실제 예측 결과를 시계열 형태로 시각화하여, 테스트 기간 전체에 걸쳐 예측값과 실제값이 얼마나 잘 일치하는지를 확인하였습니다. 이러한 시각화는 수치 지표만으로는 파악하기 어려운 예측의 정밀도와 안정성을 직관적으로 확인할 수 있도록 도와줍니다.

테스트 기간 전체에 대한 예측 결과 시각화

- 모델이 미래(Test Set)를 얼마나 잘 맞추었는지 그림으로 확인

```
# --- 전체 테스트 기간 시각화 ---
plt.figure(figsize=(15, 6))
sns.lineplot(x=y_test.index, y=y_test, label='Actual (실제값)')
sns.lineplot(x=y_test.index, y=y_pred_final, label='Predicted (예측값)')
plt.title('테스트 기간 전체 예측 결과 (실제값 vs 예측값)', fontsize=16)
plt.xlabel('Date')
plt.ylabel('Power Load')
plt.legend()
plt.show()
```

실행 결과



2. 예측 성능 시각화 - 특정 기간

기상청

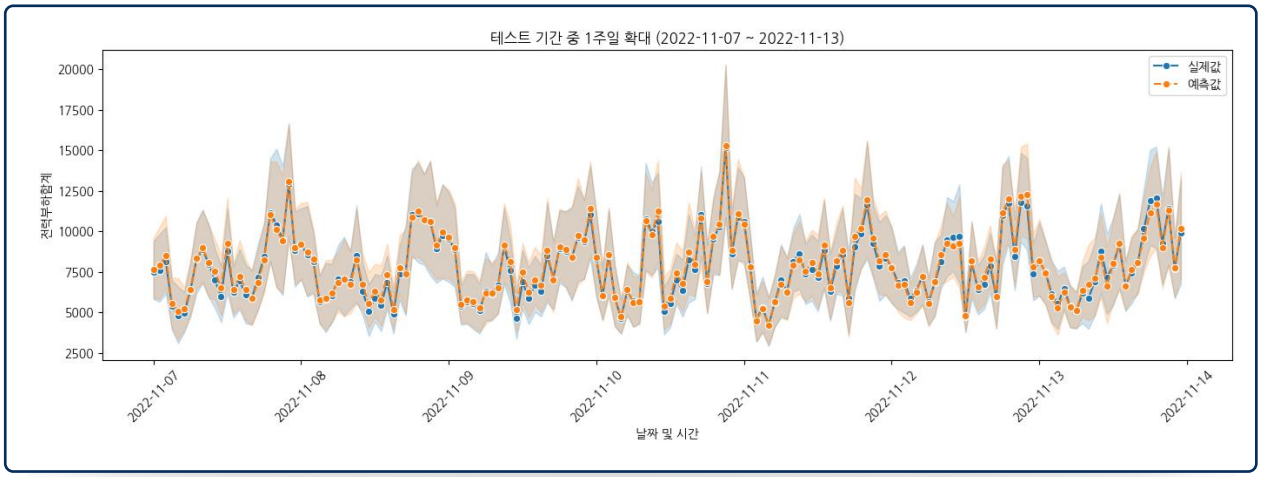
전체 기간중 임의의 1주일 구간을 확대하여 예측값과 실제값을 비교한 결과, 짧은 시간 범위에서도 모델의 예측이 실제 전력 사용 패턴을 잘 따라가고 있는 것이 확인되었습니다.
이처럼 부분 확대 시각화는 모델의 일관성과 국소적 예측 능력을 평가하는데 유의미한 정보를 제공합니다.

특정 1주일 기간에 대한 예측 결과 시각화

- 모델이 미래(Test Set)를 얼마나 잘 맞추었는지 그림으로 확인

```
# 특정 1주일 확대 시각화
sample_predictions = predictions.loc['2022-11-07':'2022-11-13']
plt.figure(figsize=(14, 5))
sns.lineplot(data=sample_predictions[['실제값', '예측값']], marker='o')
plt.title('테스트 기간 중 1주일 확대 (2022-11-07 ~ 2022-11-13)')
plt.xlabel('날짜 및 시간')
plt.ylabel('전력부하합계')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

실행 결과



3. 결론

분석 결과 종합

- 최종 모델: Random Forest Regressor
- 최종 성능: R^2(설명력) 0.96, MAPE(평균 오차율) 9.5%
- 핵심 성공 요인: 데이터의 특성(계절성, 주기성)을 깊이 이해하고 이를 모델이 학습할 수 있는 형태로 가공한 체계적인 특성 공학

프로젝트 의의

- 본 프로젝트는 공공 데이터를 활용하여, 공동주택의 미래 전력 수요를 약 7.6%의 오차율로 정확하게 예측하는 실용적인 머신러닝 모델을 성공적으로 구축했다는 점에서 큰 의의를 가짐
- 이는 데이터 기반의 과학적인 접근이 안정적인 국가 에너지망 관리와 효율화에 어떻게 실질적으로 기여할 수 있는지를 보여주는 좋은 사례

4. 향후 과제

기상청

데이터 측면: 정보의 확장

- 추가 변수 확보: 명절, 공휴일, 대체 공휴일과 같은 특수일 정보, 또는 코로나19와 같은 사회적 이벤트 변수를 추가한다면 예외적인 상황에 대한 예측 정확도를 더욱 높일 수 있음

모델 측면: 기술의 심화

- 모델의 고도화: LSTM, Prophet과 같은 딥러닝 기반의 최신 시계열 전문 모델들과 성능을 비교하여, 모델을 더욱 발전시키고 오차율을 한 자릿수 이하로 줄이는 도전을 계속할 수 있음

본 문서의 내용은 기상청 날씨마루(http://bd.kma.go.kr)의
분석 플랫폼 활용을 위한 Python 프로그래밍 교육 자료입니다.