

ポートフォリオ「電力消費量予測 可視化」(仲本 夏生)

0.0 アウトライン

この分析では、電力消費量に影響を与える要因を可視化・検証し、予測モデルの構築に役立てることを目的とする。分析を可視化とモデル化の2つのパートに分け、可視化パートでは仮説を5つ立て、それらを実データで検証していく。

可視化（本ファイル）

1. 背景・目的
2. 仮説立て
3. 仮説の検証

モデル化

1. 特徴確認
2. モデル化1 古典的時系列分析モデル
3. モデル化2 Prophetモデル
4. モデル化3 決定木系モデル
5. まとめ

1.0 背景・目的

1.1 背景

- エネルギーの需給バランスについて、特に再生可能エネルギーなどの変動制電源の導入が進む中では、安定した調整力が提供できることが重要である。そのため、需要予測の精度を向上させることは企業価値の向上に繋がると考える。
- また、機械学習の進化により、過去のデータからパターンを学習し、将来の需給を予測する能力が向上している。

1.2 目的

- 電力の消費パターンを分析することで、需給のトレンドを理解し、効果的な予測モデルを作成することを目指す。
- 本分析を通して、時系列データの扱い方や様々な産業で使われている時系列分析の手法を習得したく、多くのモデル実装を試みる。

2.0 仮説立て

2.1 分析で用いるデータセット

今回の分析で用いるデータセットは以下2種であり、いずれも一般に公開されているものとなる。

1. 東京電力パワーグリッドの電力使用実績データ（[参照](#)）
 - 対象期間：2016年4月以降
 - 公開データよりCSV形式で取得し、電力消費量の時系列データとして活用する。
2. 気象庁の気象データ（[参照](#)）
 - 地点別・項目別・期間別に取得可能なデータセット。
 - 今回は「平均気温」を1時間単位で取得し、電力消費量との関連性を分析する。

2.2 仮説立て

1. 季節による電力消費量の変動
 - 仮説：電力消費量は夏期と冬期に多く、春期と秋期には少ない。
 - 理由：夏は冷房、冬は暖房の使用量が多いことが予想される。電気代の季節ごとの変化からもこの傾向が推測できる。
2. 休日と平日の電力消費量の差
 - 仮説：土日・祝日は平日と比べて電力消費量が少ない。
 - 理由：工場などの稼働が減少する一方で、家庭での消費が増加するため、全体的には消費量が減少すると考えられる。
3. 時間帯による電力消費量の変動
 - 仮説：日中は電力消費量が多く、夜間は少なくなる。
 - 理由：工場や商業施設が稼働する日中は電力消費が多い。一方、夜間は照明や家庭での消費が増える可能性もあり、可視化で確認したい。
4. 電力逼迫警報が出された日の影響
 - 仮説：東京エリアで電力逼迫警報が出された日は消費量が通常と異なる。
 - 理由：2022年3月22日に初めて節電要請が発令された。この日が外れ値として扱うべきかを、データの推移を確認して判断する。
5. コロナ禍の影響
 - 仮説：緊急事態宣言中は電力消費量が減少する。
 - 理由：コロナ禍における経済活動の停滞やリモートワークの普及が、電力消費に影響を与える可能性がある。特に緊急事態宣言が消費量に及ぼした影響を調べる。

2.3 今後の進め方

以上の5つの仮説を基に、以下の手順で分析を進める。

- 各仮説に関連するデータを整理し、仮説を検証するためのグラフや統計指標を作成。
- 可視化結果を解釈し、仮説の妥当性を検討。
- 結論を基に、電力消費量の予測モデル構築のための特徴量を作成（モデル化パート）。

3.0 仮説の検証

3.1 データセットの読み込み

① 電力データセットの読み込みと整形

```
In [1]: # 必要なライブラリのインポート
import chardet # エンコーディングを自動的に推測する
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
from matplotlib import pyplot as plt
from PIL import Image
import warnings
warnings.simplefilter("ignore")
from matplotlib import rcParams

rcParams["font.family"] = "sans-serif"
rcParams["font.sans-serif"] = "Meiryo"
pd.set_option("display.unicode.east_asian_width", True)
```

- 2023年のデータの読み込み
 - 日ごとのcsvファイルになっているので、for文でまわして取得する。
 - 実際にcsvファイルの中身を確認し、何行目からテーブルが始まっているかを確認し、14行目～304行目を取得した（skiprows=13, skipfooter=305を引数に指定）

```
In [2]: # 2023年のデータの読み込み
import glob

# 日ごとのCSVファイルが保存されているディレクトリを指定
csv_files = glob.glob("./2023/*.csv") # CSVファイルのパスを指定
# 空のリストを用意してデータを格納
data_list = []
# 各ファイルを読み込んで結合
for file in csv_files:
    # 13~39行目をスキップ
    df = pd.read_csv(
        file, skiprows=13, skipfooter=305, encoding="MacRoman", engine="python"
    )
    # データフレームをリストに追加
    data_list.append(df)
# 全てのデータフレームを結合
tokyo_2023 = pd.concat(data_list, ignore_index=True)
# 結果を確認
print(f"2023年: {tokyo_2023.shape}")
tokyo_2023
```

2023年: (8760, 6)

```
Out[2]:
```

	DATE	TIME	電力量(kW)	電率(%)	電力密度(%)	電力密度(%)
0	2023/1/1	0:00	2870	2866	78	3672
1	2023/1/1	1:00	2721	2720	77	3492
2	2023/1/1	2:00	2636	2625	77	3385
3	2023/1/1	3:00	2575	2562	77	3333
4	2023/1/1	4:00	2548	2531	77	3303
...
8755	2023/12/31	19:00	2919	2922	70	4153
8756	2023/12/31	20:00	2845	2868	68	4151
8757	2023/12/31	21:00	2754	2766	66	4128
8758	2023/12/31	22:00	2647	2676	65	4034
8759	2023/12/31	23:00	2583	2629	65	3966

8760 rows × 6 columns

```
In [3]: # 年ごとのデータを取り込む
years = range(2016, 2023) # 2016年から2022年まで
dataframes = {}

for year in years:
    file_path = f"./juyo-{year}.csv"
    # 動的に各年のデータフレームを作成
    globals()[f"tokyo_{year}"] = pd.read_csv(file_path, encoding="MacRoman", skiprows=2)
```

```
# データの形を確認
for year in years:
    df = globals()[f"tokyo_{year}"]
    print(f"{year}年: {df.shape}")
```

```
2016年: (6600, 3)
2017年: (8760, 3)
2018年: (8760, 3)
2019年: (8760, 3)
2020年: (8784, 3)
2021年: (8760, 3)
2022年: (8760, 3)
```

```
In [4]: # 年ごとのデータフレームのカラム名を一括変更
for year in range(2016, 2023):
    globals()[f"tokyo_{year}"].rename(columns={"電力量(㎿)": "10MW"}, inplace=True)

# 特殊なケース(2023年)
tokyo_2023.rename(columns={"電力量(㎿)": "10MW"}, inplace=True)
```

```
In [5]: # 各年のDataFrameを垂直に連結する
tokyo_all = pd.concat([
    tokyo_2016,
    tokyo_2017,
    tokyo_2018,
    tokyo_2019,
    tokyo_2020,
    tokyo_2021,
    tokyo_2022,
    tokyo_2023,
])
tokyo_all.head()
```

```
Out[5]:
```

	DATE	TIME	10MW	ó¥ë™íl(ñúkW)	égopóí(%)	ääääó(ñúkW)
0	2016/4/1	0:00	2555	NaN	NaN	NaN
1	2016/4/1	1:00	2433	NaN	NaN	NaN
2	2016/4/1	2:00	2393	NaN	NaN	NaN
3	2016/4/1	3:00	2375	NaN	NaN	NaN
4	2016/4/1	4:00	2390	NaN	NaN	NaN

```
In [6]: # 日付と時間データを結合して新しい DATETIME 列を作成
tokyo_all["DATETIME"] = pd.to_datetime(tokyo_all["DATE"] + " " + tokyo_all["TIME"])
```

```
In [7]: # DATETIME 列をインデックスに設定
tokyo_all.set_index("DATETIME", inplace=True) # inplace=True(元のデータフレームをその場で変更) の場合、再代入は不要
```

```
In [8]: # 必要な列のみを選択(電力消費量)
tokyo_all = tokyo_all.loc[:, ['10MW']]
tokyo_all.head()
```

```
Out[8]:
```

	10MW
2016-04-01 00:00:00	2555
2016-04-01 01:00:00	2433
2016-04-01 02:00:00	2393
2016-04-01 03:00:00	2375
2016-04-01 04:00:00	2390

② 気温データの読み込みと整形

```
In [9]: # ファイルのエンコーディングを推測
file_path_2 = "./temperature/2016_temperature.csv"

with open(file_path_2, "rb") as f:
    rawdata = f.read()
    result = chardet.detect(rawdata)
    encoding = result["encoding"]
    print(f"推測されたエンコーディング: {encoding}")
```

推測されたエンコーディング: SHIFT_JIS

```
In [10]: # 日ごとのCSVファイルが保存されているディレクトリを指定
csv_files = glob.glob("./temperature/*.csv") # CSVファイルのパスを指定

# 空のリストを用意してデータを格納
data_list = []

# 各ファイルを読み込んで結合
for file in csv_files:
    # 13~39行目をスキップ
    df = pd.read_csv(file, skiprows=4, encoding="SHIFT_JIS", engine="python")

    # データフレームをリストに追加
```

```

data_list.append(df)

# 全てのデータフレームを結合
temperature_all = pd.concat(data_list, ignore_index=True)

# 結果を確認
temperature_all

```

Out[10]:

	年月日時	気温(°C)	気温(°C).1	気温(°C).2
0	2015/12/31 1:00	5.9	8	1
1	2015/12/31 2:00	4.6	8	1
2	2015/12/31 3:00	4.3	8	1
3	2015/12/31 4:00	3.6	8	1
4	2015/12/31 5:00	3.6	8	1
...
70147	2023/12/31 20:00	11.4	8	1
70148	2023/12/31 21:00	11.0	8	1
70149	2023/12/31 22:00	9.5	8	1
70150	2023/12/31 23:00	9.4	8	1
70151	2024/1/1 0:00	9.1	8	1

70152 rows × 4 columns

In [11]:

```

# 2列目と3列目を削除
temperature_all = temperature_all.drop(columns=temperature_all.columns[[2, 3]])

# 1から23行目を削除 (0-based indexの0~22行目)
temperature_all = temperature_all.drop(index=temperature_all.index[0:23])

# インデックスを振り直し、元のインデックスは削除
temperature_all = temperature_all.reset_index(drop=True)

temperature_all.head()

```

Out[11]:

	年月日時	気温(°C)
0	2016/1/1 0:00	5.2
1	2016/1/1 1:00	5.2
2	2016/1/1 2:00	5.0
3	2016/1/1 3:00	4.0
4	2016/1/1 4:00	3.9

In [12]:

```

# 日付と時間データを結合して新しい DATETIME 列を作成
temperature_all["DATETIME"] = pd.to_datetime(temperature_all["年月日時"])
# DATETIME 列をインデックスに設定
temperature_all.set_index("DATETIME", inplace=True) # inplace=True(元のデータフレームをその場で変更) の場合、再代入は不要
temperature_all = temperature_all.drop(columns=temperature_all.columns[0])
temperature_all = temperature_all.rename(columns={"気温(°C)": "temperature"})
temperature_all.head()

```

Out[12]:

	temperature
DATETIME	
2016-01-01 00:00:00	5.2
2016-01-01 01:00:00	5.2
2016-01-01 02:00:00	5.0
2016-01-01 03:00:00	4.0
2016-01-01 04:00:00	3.9

③ データの結合

In [13]:

```

# 気温データをDATETIMEで電力データと結合
merged_data = pd.merge(tokyo_all, temperature_all, on="DATETIME", how="left")
merged_data.head()

```

Out[13]:

	10MW	temperature
DATETIME		
2016-04-01 00:00:00	2555	14.5
2016-04-01 01:00:00	2433	13.3
2016-04-01 02:00:00	2393	13.8
2016-04-01 03:00:00	2375	13.0
2016-04-01 04:00:00	2390	12.2

3.2 可視化を通して仮説検証

前提 全期間で数値要約と可視化

- 細分化して分析する前に全期間での数値要約と可視化を通じて、データの概形を掴みたい

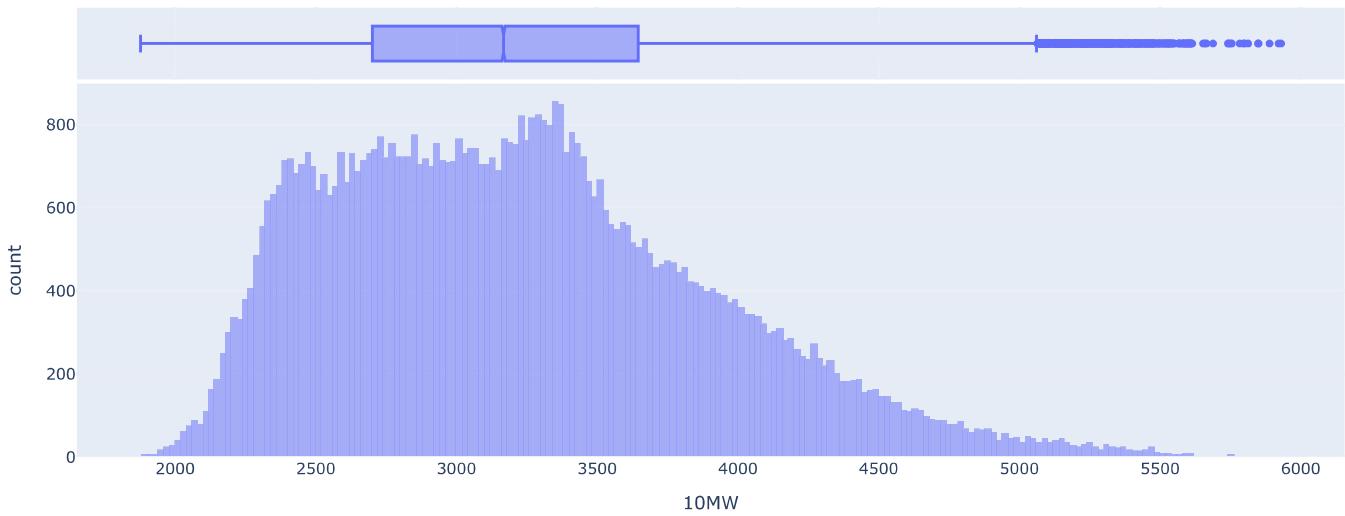
```
In [14]: # データの基本統計量を調べる  
merged_data.describe()
```

```
Out[14]: 10MW      temperature
```

	10MW	temperature
count	67944.000000	67940.000000
mean	3226.734929	16.911391
std	669.197716	8.235110
min	1877.000000	-3.600000
25%	2701.000000	10.000000
50%	3167.000000	17.300000
75%	3646.000000	23.600000
max	5930.000000	37.400000

```
In [15]: # データの分布を確認する  
fig = px.histogram(merged_data, x="10MW", barmode="overlay", marginal="box", title="電力消費量の分布")  
fig.show()
```

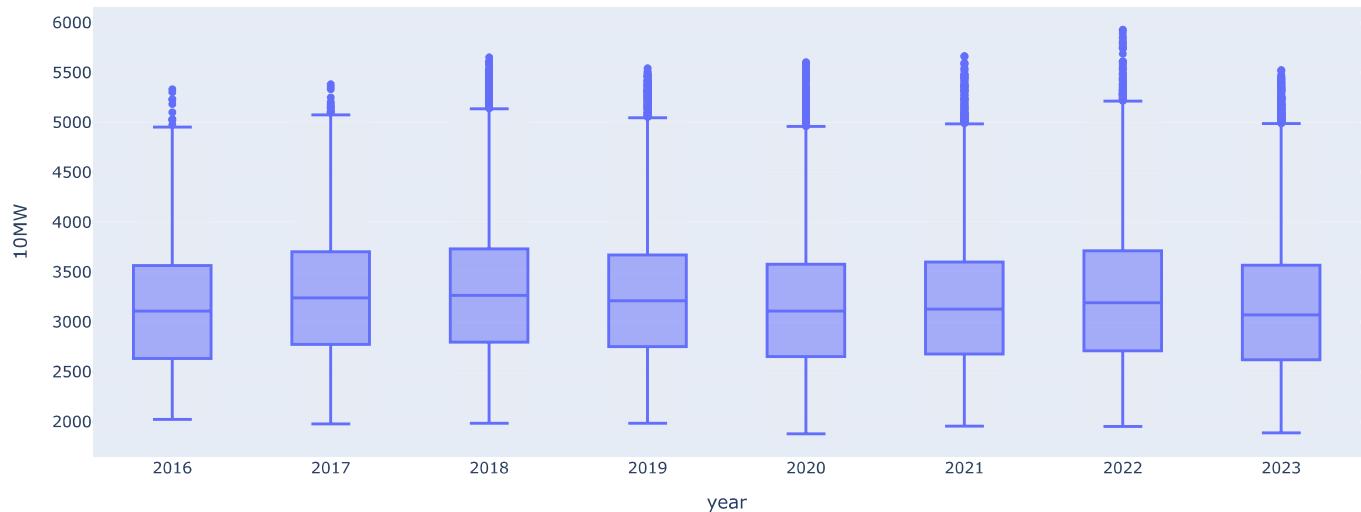
電力消費量の分布



- 中央値は317万kW、最小値は188万kW、最大値は593万kWであり、棒グラフで見ると左に偏った分布になる
- 電力使用量が多くなることは件数としては比較的少なく、おおよそ240~350万kWの消費量が多いことが分かる

```
In [16]: # 年ごとの電力使用量の箱ひげ図を確認  
merged_data["year"] = merged_data.index.year  
fig = px.box(merged_data, x="year", y="10MW", title="年ごとの箱ひげ図")  
fig.show()
```

年ごとの箱ひげ図

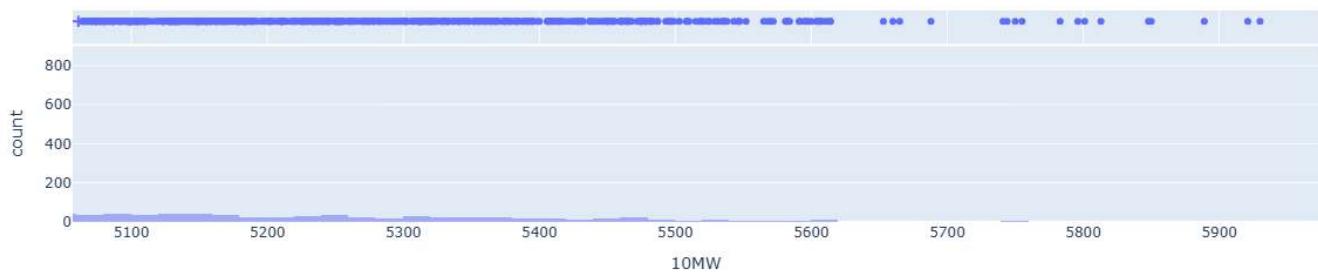


- 外れ値の可能性を考慮するために、先ほどのヒストグラムは右裾を抜き出した

```
In [17]: filename = "./image/image_1.png"
im = Image.open(filename)
im
```

Out[17]:

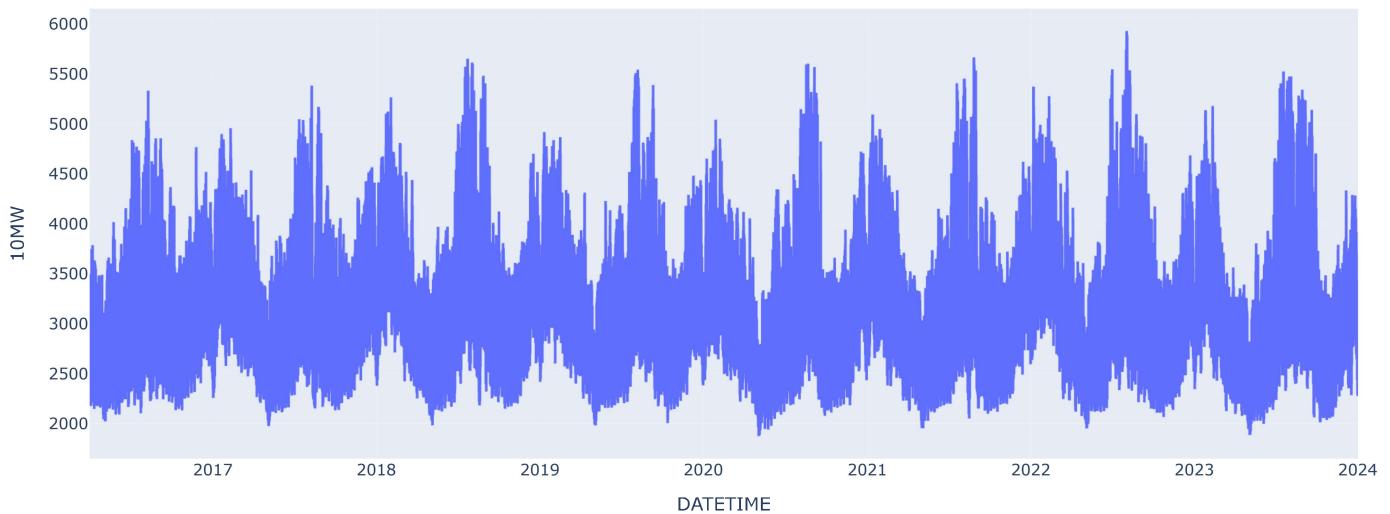
電力消費量の分布



- このヒストグラムを見る限り、右裾野の電力消費量が高い値に集中しているわけではなく、なだらかに推移していることが確認できる。
- したがって、明確な外れ値（異常値）は存在しないと判断しても良いと考えられる。
- ただし、イレギュラーな条件（天候や電力供給に関する特殊要因）が影響している可能性があるため、これについては仮説4（電力需給ひつ迫警報）でも検証してみたい。

```
In [18]: # 全期間の推移をみるために時系列グラフを描画
fig = px.line(
    merged_data, x=merged_data.index, y="10MW", title="電力消費量の推移（全期間）")
fig.show()
```

電力消費量の推移（全期間）



- 長期的な上昇/下降トレンドは見られない。
- 一年ごとの周期性が見られる（夏と冬にピーク）。

仮説1. 季節による電力消費量の変動

- 仮説：電力消費量は夏期と冬期に多く、春期と秋期には少ない。
- 手法：月ごとの電力消費量の平均を折れ線グラフで可視化

```
In [19]: # インデックスから時間のカラムを追加
merged_data["month"] = merged_data.index.month
merged_data["hour"] = merged_data.index.hour
merged_data["week"] = merged_data.index.weekday
merged_data.head()
```

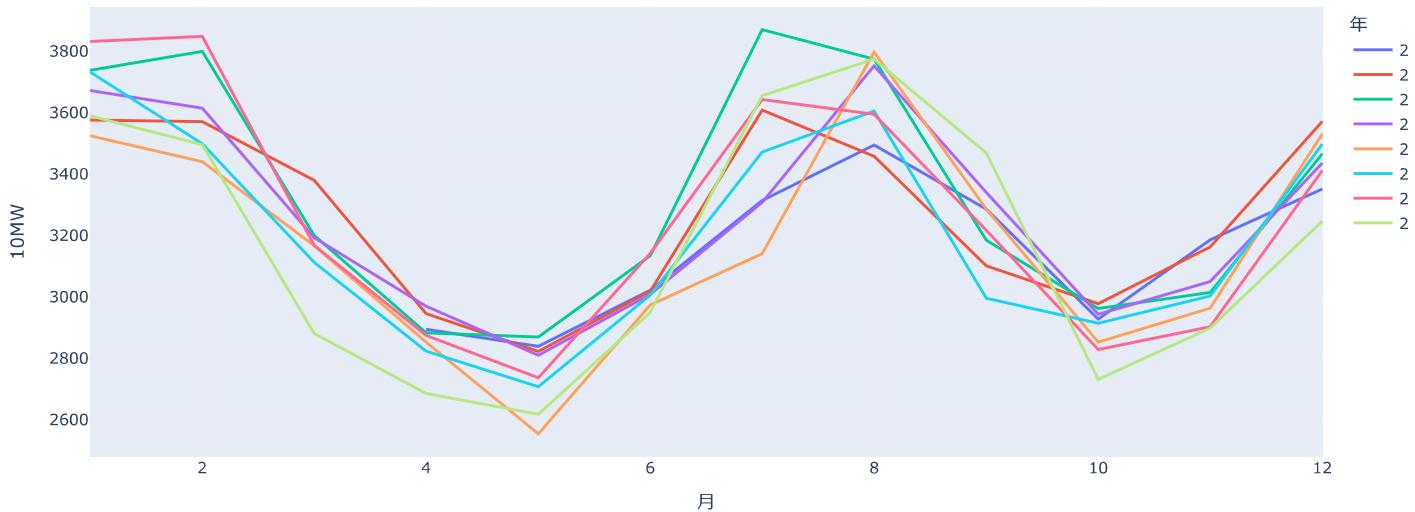
```
Out[19]:      10MW  temperature  year  month  hour  week
DATETIME
2016-04-01 00:00:00    2555       14.5  2016      4     0     4
2016-04-01 01:00:00    2433       13.3  2016      4     1     4
2016-04-01 02:00:00    2393       13.8  2016      4     2     4
2016-04-01 03:00:00    2375       13.0  2016      4     3     4
2016-04-01 04:00:00    2390       12.2  2016      4     4     4
```

```
In [20]: # 年ごとの平均を月ごとに集計する
yearly_avg = merged_data.groupby(["year", "month"])[["10MW"]].mean().reset_index()
# groupby()でグループ化された値は通常インデックスに格納されるが、reset_index()をすることで 'month' と 'hour' が通常の列になる
yearly_avg.head().groupby(["year", "month"])[["10MW"]].mean().reset_index()
# groupby()でグループ化された値は通常インデックスに格納されるが、reset_index()をすることで 'month' と 'hour' が通常の列になる
yearly_avg.head()
```

```
Out[20]:   year  month      10MW
0  2016      4  2893.994444
1  2016      5  2838.592742
2  2016      6  3021.763889
3  2016      7  3314.485215
4  2016      8  3494.958333
```

```
In [21]: # 年別の時系列推移
fig = px.line(
    yearly_avg,
    x="month",
    y="10MW",
    color="year",
    title="平均電力消費量の時系列推移（年別）",
)
# x軸を時間として設定
fig.update_xaxes(title_text="月", tickformat="%M")
fig.update_layout(legend_title_text="年")
fig.show()
```

平均電力消費量の時系列推移（年別）



- どの年も夏期（7～8月）と冬期（1～2月）に消費量がピークに達する。
- 春と秋の消費量は比較的低い。
- また、この原因が気温の上昇と下降にあると考え、気温データと共に比較してみたい

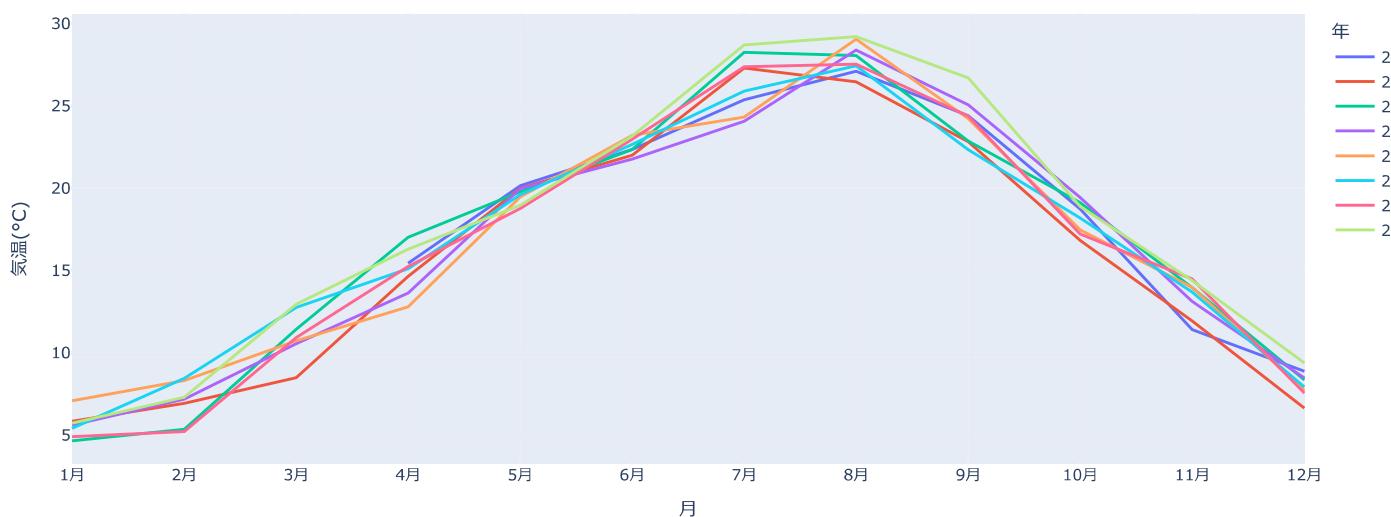
```
In [22]: # 年ごとの平均を月ごとに集計する
monthly_avg = (
    merged_data.groupby(["year", "month"])[["10MW", "temperature"]].mean().reset_index()
)
monthly_avg.head()
```

```
Out[22]:   year  month      10MW  temperature
0  2016      4  2893.994444  15.437500
1  2016      5  2838.592742  20.149194
2  2016      6  3021.763889  22.348750
3  2016      7  3314.485215  25.370699
4  2016      8  3494.958333  27.113172
```

```
In [23]: # 月ごとの平均気温を可視化
fig = px.line(
    monthly_avg,
    x="month",
    y="temperature",
    color="year",
    title="平均気温の時系列推移（年別）",
    labels={"month": "月", "temperature": "気温(°C)", "year": "年"},
)

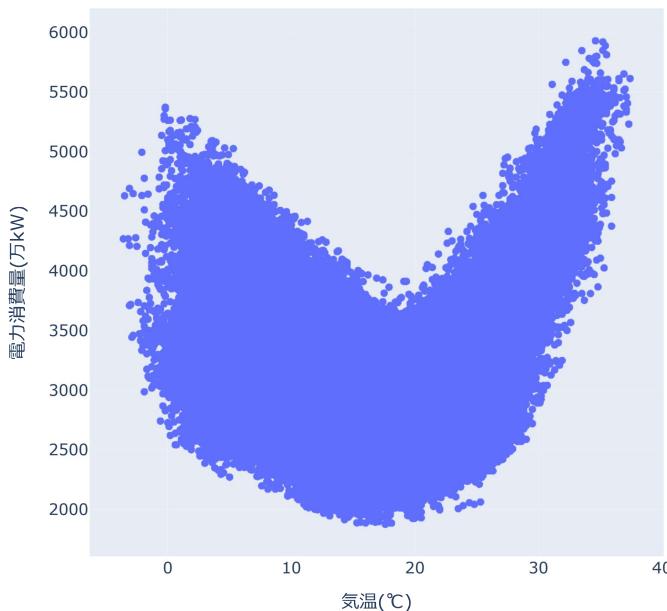
# x軸の設定：1月から12月までを表示
fig.update_xaxes(
    tickmode="array",
    tickvals=list(range(1, 13)),
    ticktext=[
        "1月",
        "2月",
        "3月",
        "4月",
        "5月",
        "6月",
        "7月",
        "8月",
        "9月",
        "10月",
        "11月",
        "12月",
    ],
)
# グラフを表示
fig.show()
```

平均気温の時系列推移（年別）



```
In [24]: # 気温と電力消費量の関係を散布図で可視化
fig = px.scatter(
    merged_data,
    x="temperature",
    y="10MW",
    title="気温と電力消費量の関係",
    labels={"temperature": "気温(°C)", "10MW": "電力消費量(万kW)"})
# グラフを正方形に設定
fig.update_layout(
    width=600, # 正方形の幅
    height=600, # 正方形の高さ
)
fig.show()
```

気温と電力消費量の関係



```
In [25]: # 気温と電力消費量の相関係数
correlation = merged_data[["temperature", "10MW"]].corr()
print(f"気温と電力消費量の相関係数:\n{correlation}")
```

気温と電力消費量の相関係数:

	temperature	10MW
temperature	1.000000	0.092387
10MW	0.092387	1.000000

- 夏と冬に電力消費量が増加するのは、気温の高低による冷房・暖房の使用量増加が要因と考えられる。
- 散布図から、気温と電力消費量の関係は非線形で凸型の傾向が見られる。
- 特微量エンジニアリングで非線形性を考慮する方法を次のモデル化パートで検討する。

仮説2. 休日と平日の電力消費量の差

- 仮説：土日・祝日は平日と比べて電力消費量が少ない。
- 手法：曜日ごとの電力消費量の平均を折れ線グラフで可視化

```
In [26]: # 曜日データを日本語の曜日に変換
week_mapping = {0: "月", 1: "火", 2: "水", 3: "木", 4: "金", 5: "土", 6: "日"}
merged_data["week_jp"] = merged_data["week"].map(week_mapping)
```

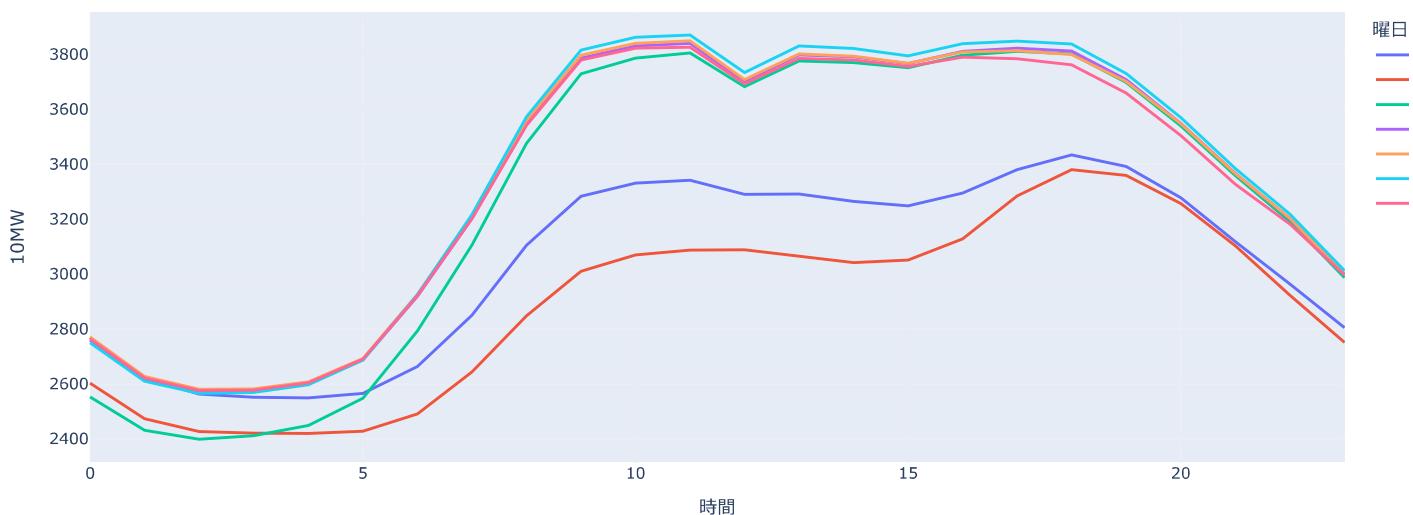
```
In [27]: # 曜日別で電力消費量の違いを比べる
# インデックスを使って曜日ごとに集計
weekly_avg = merged_data.groupby(["week_jp", "hour"])["10MW"].mean().reset_index()
weekly_avg.head()
```

```
Out[27]:   week_jp  hour      10MW
0    土       0  2767.911111
1    土       1  2620.750617
2    土       2  2563.019753
3    土       3  2551.012346
4    土       4  2548.837037
```

```
In [28]: # 曜日ごとの1日の平均電力消費量をプロット
fig = px.line(
    weekly_avg,
    x="hour",
    y="10MW",
    color="week_jp",
    title="1日の中の平均電力消費量の1時間ごとの推移（曜日別）",
)
# x軸を時間として設定
fig.update_xaxes(
    title_text="時間", tickformat="%H"
) # 24時間制で「時」を表示するフォーマット

fig.update_layout(legend_title_text="曜日")
fig.show()
```

1日の中の平均電力消費量の1時間ごとの推移（曜日別）



- 平日（月～金）は消費量が高い。
- 土曜と日曜は明らかに消費量が低下している。
- 一方で、この曜日ラベルは祝日を考慮できていないので、ここから祝日要素を取り入れていきたい。

【年固定の日にある祝日】

- 元日: 1月1日 ※年末年始12月29日～1月3日は本来祝日ではないが多くの企業が休みに入るため、今回は祝日に含める
- 建国記念の日: 2月11日
- 昭和の日: 4月29日
- 憲法記念日: 5月3日
- みどりの日: 5月4日
- こどもの日: 5月5日
- 文化の日: 11月3日
- 勤労感謝の日: 11月23日

```
In [29]: from datetime import datetime, timedelta
```

```
# 固定祝日リスト（毎年同じ日）
def get_fixed_holidays(year):
    return [
        f'{year}-12-29', # 年末年始
        f'{year}-12-30', # 年末年始
        f'{year}-12-31', # 年末年始
        f'{year}-01-01', # 元日（祝日）
        f'{year}-01-02', # 年末年始
        f'{year}-01-03', # 年末年始
        f'{year}-02-11', # 建国記念の日
        f'{year}-04-29', # 昭和の日
        f'{year}-05-03', # 憲法記念日
        f'{year}-05-04', # みどりの日
        f'{year}-05-05', # こどもの日
        f'{year}-08-11', # 山の日
        f'{year}-11-03', # 文化の日
        f'{year}-11-23', # 勤労感謝の日
    ]
```

```
# 年ごとの変動祝日リスト
def get_variable_holidays(year):
    holidays = []

    # 成人の日（1月第2月曜日）
    holidays.append(
        pd.date_range(start=f'{year}-01-01', end=f'{year}-01-31', freq='W-MON')[1]
    )

    # 海の日（7月第3月曜日）※2020年は7月23日、2021年は7月22日に変更
    if year == 2020:
        holidays.append(pd.to_datetime(f'{year}-07-23'))
    elif year == 2021:
        holidays.append(pd.to_datetime(f'{year}-07-22'))
    else:
        holidays.append(
            pd.date_range(start=f'{year}-07-01', end=f'{year}-07-31', freq='W-MON')[2]
        )

    # 敬老の日（9月第3月曜日）
    holidays.append(
        pd.date_range(start=f'{year}-09-01', end=f'{year}-09-30', freq='W-MON')[2]
    )

    # スポーツの日（10月第2月曜日）※2020年、2021年は変更
    if year == 2020:
        holidays.append(pd.to_datetime(f'{year}-07-24'))
    elif year == 2021:
        holidays.append(pd.to_datetime(f'{year}-07-23'))
    else:
        holidays.append(
            pd.date_range(start=f'{year}-10-01', end=f'{year}-10-31', freq='W-MON')[1]
        )

    # 秋分の日（年によって異なる）
    if year == 2016:
        holidays.append(pd.to_datetime(f'{year}-09-22'))
    elif year == 2017:
        holidays.append(pd.to_datetime(f'{year}-09-23'))
    elif year == 2018:
        holidays.append(pd.to_datetime(f'{year}-09-23'))
    elif year == 2019:
        holidays.append(pd.to_datetime(f'{year}-09-23'))
    elif year == 2020:
        holidays.append(pd.to_datetime(f'{year}-09-22'))
    elif year == 2021:
        holidays.append(pd.to_datetime(f'{year}-09-23'))
    elif year == 2022:
        holidays.append(pd.to_datetime(f'{year}-09-23'))
    elif year == 2023:
        holidays.append(pd.to_datetime(f'{year}-09-23"))

    return holidays
```

```
# 振替休日を追加する関数
def add_substitute_holiday(holidays):
    holidays_with_substitute = holidays.copy() # 祝日リストのコピーを作成
    for holiday in holidays:
        holiday_date = pd.to_datetime(holiday) # 日付をdatetime型に変換
        # もし祝日が日曜日（weekday() == 6）なら、次の月曜日に振替休日を追加
        if holiday_date.weekday() == 6: # 日曜日かどうかをチェック
            substitute_date = holiday_date + timedelta(
                days=1
            ) # 翌日（月曜日）を振替休日とする
            # timedeltaオブジェクトを使用して、日付や時間の差分を表す
            holidays_with_substitute.append(
                substitute_date.strftime("%Y-%m-%d")
            ) # 振替休日を追加
    return holidays_with_substitute
```

```
In [30]: # 2016年から2023年までの祝日を取得
def get_holidays_for_years(start_year, end_year):
    all_holidays = []
    for year in range(start_year, end_year + 1):
```

```

# 固定祝日と変動祝日を取得
fixed_holidays = get_fixed_holidays(year)
variable_holidays = get_variable_holidays(year)

# 日付が文字列として処理されないように、datetimeに変換してから文字列化
all_holidays_for_year = fixed_holidays + [
    pd.to_datetime(date).strftime("%Y-%m-%d") for date in variable_holidays
]
# **strftime**は、日付を文字列形式に変換するメソッドで、「%Y-%m-%d」というフォーマットを使って日付を整形

# 振替休日を追加
all_holidays_for_year = add_substitute_holiday(all_holidays_for_year)
all_holidays.extend(all_holidays_for_year)
# extend()は、リストをそのまま追加するのではなく、そのリストの中身を要素として追加

return sorted(all_holidays)

```

```
In [31]: # 祝日リスト（例：2016年～2023年の祝日）
all_holidays = get_holidays_for_years(2016, 2023)
# 祝日をdatetime形式に変換
all_holidays = pd.to_datetime(all_holidays)
all_holidays
```

```
Out[31]: DatetimeIndex(['2016-01-01', '2016-01-02', '2016-01-03', '2016-01-04',
       '2016-01-11', '2016-02-11', '2016-04-29', '2016-05-03',
       '2016-05-04', '2016-05-05',
       ...
       '2023-08-11', '2023-09-18', '2023-09-23', '2023-10-09',
       '2023-11-03', '2023-11-23', '2023-12-29', '2023-12-30',
       '2023-12-31', '2024-01-01'],
      dtype='datetime64[ns]', length=168, freq=None)
```

```
In [32]: # インデックスのdate部分を取得して祝日かどうか判定（時間を無視して日付のみを操作するnormalizeメソッド）
# is_holiday列には1または0を使いため、astype(int)で型変換を行う
merged_data["is_holiday"] = merged_data.index.normalize().isin(all_holidays).astype(int)

# week列から土曜・日曜にフラグをつける
# 'week' が 5（土曜日）または 6（日曜日）の行に is_holiday を 1 に設定
merged_data.loc[merged_data["week"].isin([5, 6]), "is_holiday"] = 1

# 判定後のデータを確認
is_holiday_1 = merged_data[merged_data["is_holiday"] == 1]
is_holiday_1.head()
```

```
Out[32]:
          10MW  temperature  year  month  hour  week  week_jp  is_holiday
DATETIME
2016-04-02 00:00:00  2613        8.4  2016      4    0     5    土     1
2016-04-02 01:00:00  2492        8.2  2016      4    1     5    土     1
2016-04-02 02:00:00  2431        8.1  2016      4    2     5    土     1
2016-04-02 03:00:00  2403        8.4  2016      4    3     5    土     1
2016-04-02 04:00:00  2390        8.8  2016      4    4     5    土     1
```

```
In [33]: # 平日かつ祝日である日に7というフラグを立てる
merged_data.loc[(merged_data["week"] < 5) & (merged_data["is_holiday"] > 0), "week"] = 7
```

```
In [34]: # 既存の week_jp マッピングに 7 の場合のテキストを追加
weekday_mapping = {
    0: "月",
    1: "火",
    2: "水",
    3: "木",
    4: "金",
    5: "土",
    6: "日",
    7: "平日が休みの祝日",
}

# week_jp 列を更新する（weekの値に基づいて日本語の曜日を付与）
merged_data["week_jp"] = merged_data["week"].map(weekday_mapping)

result = merged_data[merged_data["week_jp"] == "平日が休みの祝日"]
result.head()
```

```
Out[34]:
          10MW  temperature  year  month  hour  week  week_jp  is_holiday
DATETIME
2016-04-29 00:00:00  2483        14.4  2016      4    0     7 平日が休みの祝日     1
2016-04-29 01:00:00  2364        14.5  2016      4    1     7 平日が休みの祝日     1
2016-04-29 02:00:00  2334        14.4  2016      4    2     7 平日が休みの祝日     1
2016-04-29 03:00:00  2341        14.1  2016      4    3     7 平日が休みの祝日     1
2016-04-29 04:00:00  2340        15.5  2016      4    4     7 平日が休みの祝日     1
```

```
In [35]: # 曜日別で電力消費量の違いを比べる
# インデックスを使って曜日ごとに集計
weekly_avg = merged_data.groupby(["week_jp", "hour"])[["10MW"]].mean().reset_index()
weekly_avg
```

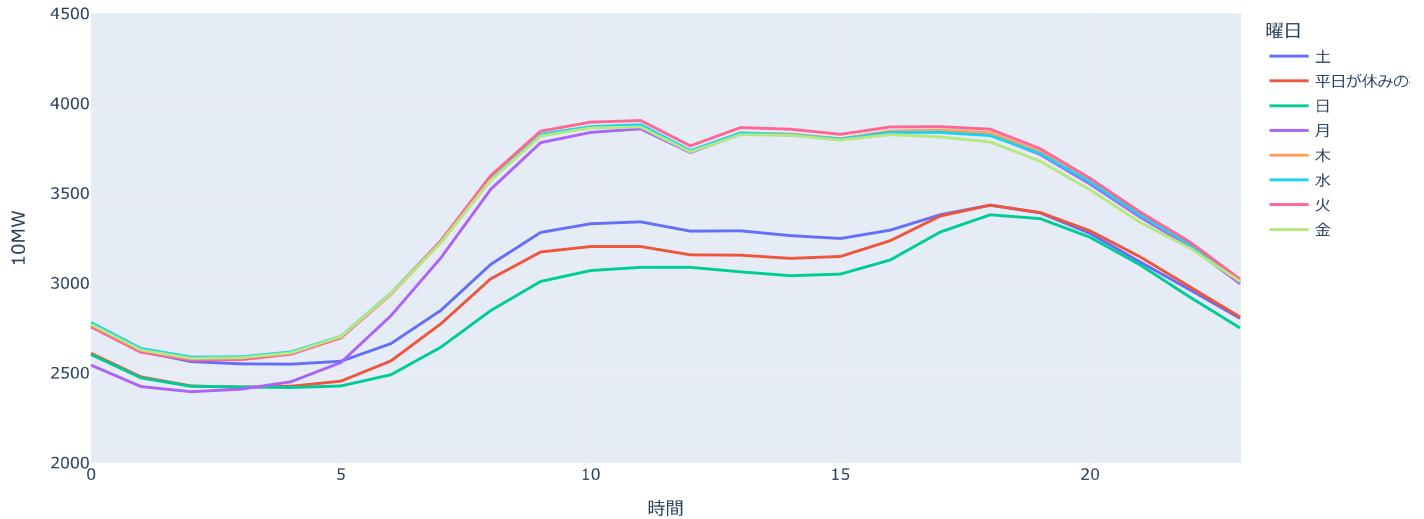
	week_jp	hour	10MW
0	土	0	2767.911111
1	土	1	2620.750617
2	土	2	2563.019753
3	土	3	2551.012346
4	土	4	2548.837037
...
187	金	19	3679.117493
188	金	20	3519.767624
189	金	21	3339.765013
190	金	22	3195.885117
191	金	23	3008.890339

192 rows × 3 columns

```
In [36]: # 曜日ごとの1日の平均電力消費量をプロット
fig = px.line(
    weekly_avg,
    x="hour",
    y="10MW",
    color="week_jp",
    title="1日の中の平均電力消費量の1時間ごとの推移（曜日別）",
)

# x軸を時間として設定
fig.update_xaxes(
    title_text="時間", tickformat="%H"
) # 24時間制で「時」を表示するフォーマット
# y軸の範囲を2500以下も表示されるように設定（例: 2300～4500に設定）
fig.update_yaxes(range=[2000, 4500])
fig.update_layout(legend_title_text="曜日")
fig.show()
```

1日の中の平均電力消費量の1時間ごとの推移（曜日別）



- 平日が休みの祝日も土曜、日曜と同じ推移をたどる
- よって、平日か祝日も含めた休日かで電力消費量が違うことが分かった。

仮説3. 時間帯による電力消費量の変動

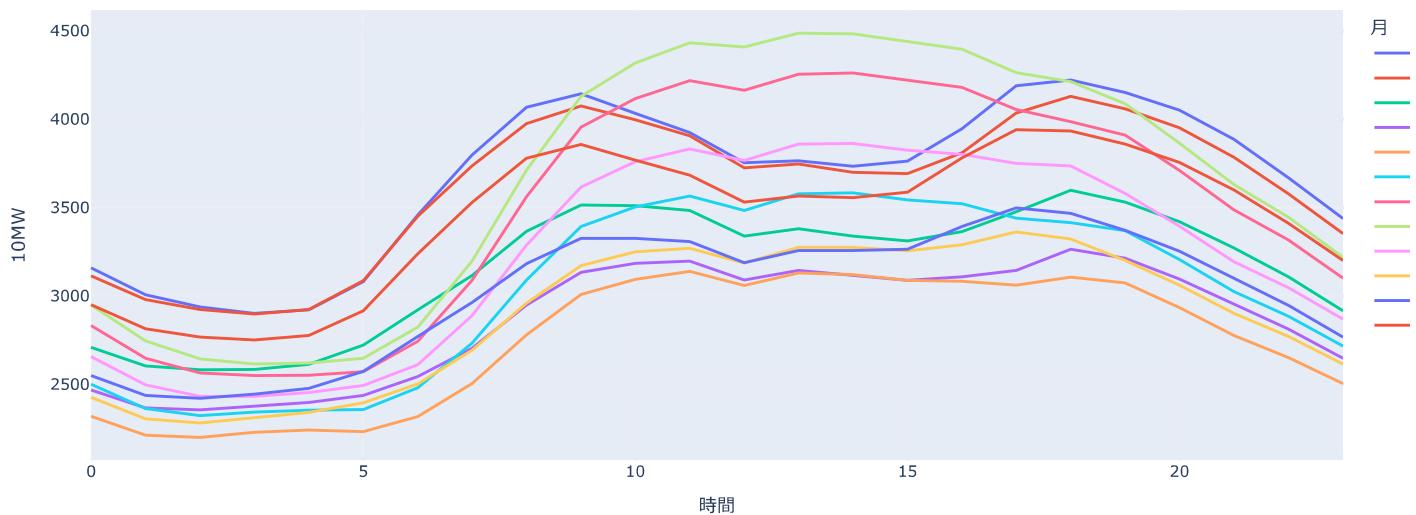
- 仮説：日中は電力消費量が多く、夜間は少なくなる。
- 手法：月ごとに一日の電力消費量の平均を折れ線グラフで可視化

```
In [37]: # 1時間ごとの平均を月ごとに集計する
monthly_avg = merged_data.groupby(["month", "hour"])["10MW"].mean().reset_index()
# groupby()でグループ化された値は通常インデックスに格納されるが、reset_index()をすることで 'month' と 'hour' が通常の列になる
monthly_avg.head()
```

```
Out[37]:   month  hour      10MW
0       1     0  3158.327189
1       1     1  3007.327189
2       1     2  2937.465438
3       1     3  2901.585253
4       1     4  2921.465438
```

```
In [38]: # 月別の時系列推移
fig = px.line(
    monthly_avg,
    x="hour",
    y="10MW",
    color="month",
    title="1日の中の平均電力消費量の推移（月別）",
)
# x軸を時間として設定
fig.update_xaxes(title_text="時間", tickformat="%H")
fig.update_layout(legend_title_text="月")
fig.show()
```

1日の中の平均電力消費量の推移（月別）



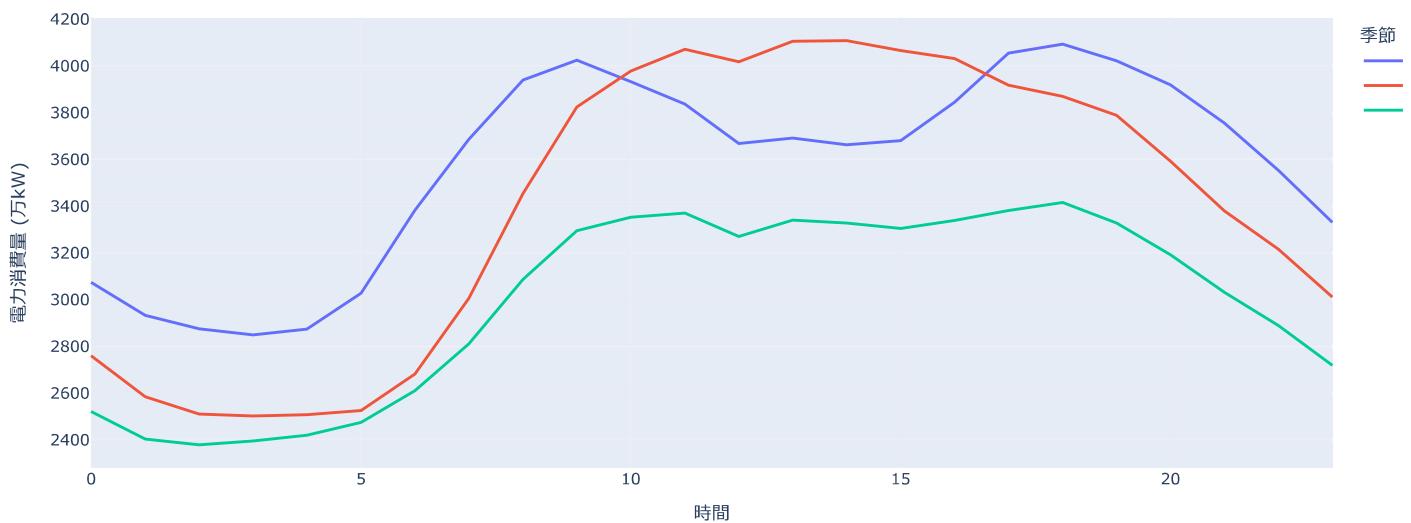
- 夏は日中に盛り上がるような形になっている。
- 12月、1月、2月は山が10時前と17時頃に二つあるような形になっている。
- この傾向をより分かりやすくするために、季節別にプロットしてみる。

```
In [39]: # 季節で色分けのルールを設定
def categorize_season(month):
    if month in [12, 1, 2]: # 冬
        return '冬'
    elif month in [6, 7, 8]: # 夏
        return '夏'
    else: # 春秋
        return '春秋'

monthly_avg["season"] = monthly_avg["month"].apply(categorize_season)
season_hour_avg = monthly_avg.groupby(['season', 'hour'])['10MW'].mean().reset_index()

# 季節別の色分けでグラフ作成
fig = px.line(
    season_hour_avg,
    x="hour",
    y="10MW",
    color="season",
    title="季節別の1日の平均電力消費量の推移",
    labels={"hour": "時間", "10MW": "電力消費量 (万kW)", "season": "季節"}
)
fig.show()
```

季節別の1日の平均電力消費量の推移



- このグラフから以下が読み取れる
 - 夏は日中に盛り上がる形
 - 冬は二方性（朝と夕方のピーク）
 - 春秋は日中も平坦な形（気温が快適なために冷暖房の使用が少なくなる）
- 以上より、季節ごとの違いはあるが、一年を通して、日中に電力消費量が多くなり、夕方になるにつれ少なくなる傾向にあることが分かった。

仮説4. 電力ひつ迫警報が出された日の影響

- 仮説：東京エリアで電力ひつ迫警報が出された日（2022年3月22日）は消費量が通常と異なる。
- 手法：該当日と2022年3月平日の平均電力消費量を折れ線グラフで可視化する

補足：電力ひつ迫警報発令の詳細

- 需給ひつ迫警報：
 - 政府は3月21日（月・祝）20時、「需給ひつ迫警報（第1報）」を東京エリアに発令。
 - 22日の東北エリアでも需給ひつ迫の見通しとなり、「需給ひつ迫警報（第2報告）」で東北エリアも対象に追加。

経緯

- 火力発電所の停止：
 - 3月16日の福島県沖地震により14基の火力発電所（約647.9万kW）が停止。
 - 3月22日時点でも6基（約334.7万kW）が停止したままだった。
- 気象条件：
 - 22日の東日本は悪天候で、日中の気温が平年より大幅に低下。
 - 電力需要がこの時期として異例の高水準となった。
- 再生可能エネルギーの供給不足：悪天候の影響で太陽光発電の供給力が伸びなかった。
- 計画外停止：連休中に磯子火力発電所（横浜市）などの追加の計画外停止が発生し、供給力がさらに減少。
- 電力需給の状況：
 - 東京エリアの需要見通し：
 - 19日（土）20時時点では4,300万kW。
 - 21日（月・祝）17時時点では4,840万kWと大幅増加。
 - これは10年で一度の厳しい寒さを想定した3月の最大需要を約300万kW上回る水準。
- 参考：
 - 2022年3月の東日本における電力需給ひつ迫に係る検証について（資源エネルギー庁）
 - 2022年3月の東日本における電力需給ひつ迫に係る検証 取りまとめ（案）（電力・ガス基本政策小委員会）

```
In [40]: # 2022年3月の平日データを抽出 (weekから4: 月～金)
march_weekdays = merged_data[
    (merged_data.index.year == 2022) & (merged_data.index.month == 3) & (tokyo_all.index.weekday < 5)
]

# 3月の平日平均を時間ごとに計算
march_weekday_avg = (
    march_weekdays.groupby(march_weekdays.index.time)[["10MW"]].mean().reset_index()
)
march_weekday_avg.columns = ["time", "10MW"]
march_weekday_avg.head()
```

```
Out[40]:    time      10MW
0 00:00:00 2695.086957
1 01:00:00 2607.565217
2 02:00:00 2600.652174
3 03:00:00 2615.521739
4 04:00:00 2664.347826
```

```
In [41]: # 2022年3月22日のデータをフィルタリング
march_22_data = merged_data[
    merged_data.index.date == pd.to_datetime("2022-03-22").date()
]
# 時間部分 (time型) を抽出
march_22_data["time"] = march_22_data.index.time
march_22_data
```

```
Out[41]:   10MW  temperature  year  month  hour  week  week_jp  is_holiday  time
DATETIME
2022-03-22 00:00:00  2587     10.0  2022      3    0     1    火    0  00:00:00
2022-03-22 01:00:00  2505      8.8  2022      3    1     1    火    0  01:00:00
2022-03-22 02:00:00  2511      8.1  2022      3    2     1    火    0  02:00:00
2022-03-22 03:00:00  2539      6.9  2022      3    3     1    火    0  03:00:00
2022-03-22 04:00:00  2608      5.6  2022      3    4     1    火    0  04:00:00
2022-03-22 05:00:00  2781      5.3  2022      3    5     1    火    0  05:00:00
2022-03-22 06:00:00  3115      4.9  2022      3    6     1    火    0  06:00:00
2022-03-22 07:00:00  3516      4.6  2022      3    7     1    火    0  07:00:00
2022-03-22 08:00:00  4005      4.0  2022      3    8     1    火    0  08:00:00
2022-03-22 09:00:00  4358      3.9  2022      3    9     1    火    0  09:00:00
2022-03-22 10:00:00  4461      3.3  2022      3   10     1    火    0  10:00:00
2022-03-22 11:00:00  4518      3.4  2022      3   11     1    火    0  11:00:00
2022-03-22 12:00:00  4436      1.9  2022      3   12     1    火    0  12:00:00
2022-03-22 13:00:00  4534      1.9  2022      3   13     1    火    0  13:00:00
2022-03-22 14:00:00  4514      2.5  2022      3   14     1    火    0  14:00:00
2022-03-22 15:00:00  4439      2.2  2022      3   15     1    火    0  15:00:00
2022-03-22 16:00:00  4377      2.1  2022      3   16     1    火    0  16:00:00
2022-03-22 17:00:00  4319      2.3  2022      3   17     1    火    0  17:00:00
2022-03-22 18:00:00  4308      2.2  2022      3   18     1    火    0  18:00:00
2022-03-22 19:00:00  4154      2.4  2022      3   19     1    火    0  19:00:00
2022-03-22 20:00:00  3962      2.7  2022      3   20     1    火    0  20:00:00
2022-03-22 21:00:00  3749      2.7  2022      3   21     1    火    0  21:00:00
2022-03-22 22:00:00  3538      3.0  2022      3   22     1    火    0  22:00:00
2022-03-22 23:00:00  3311      2.1  2022      3   23     1    火    0  23:00:00
```

```
In [42]: # グラフオブジェクトを使用してプロット
fig = go.Figure()

# 3月22日のデータをプロット
fig.add_trace(
    go.Scatter(
        x=march_22_data["time"], y=march_22_data["10MW"], mode="lines", name="3月22日"
    )
)

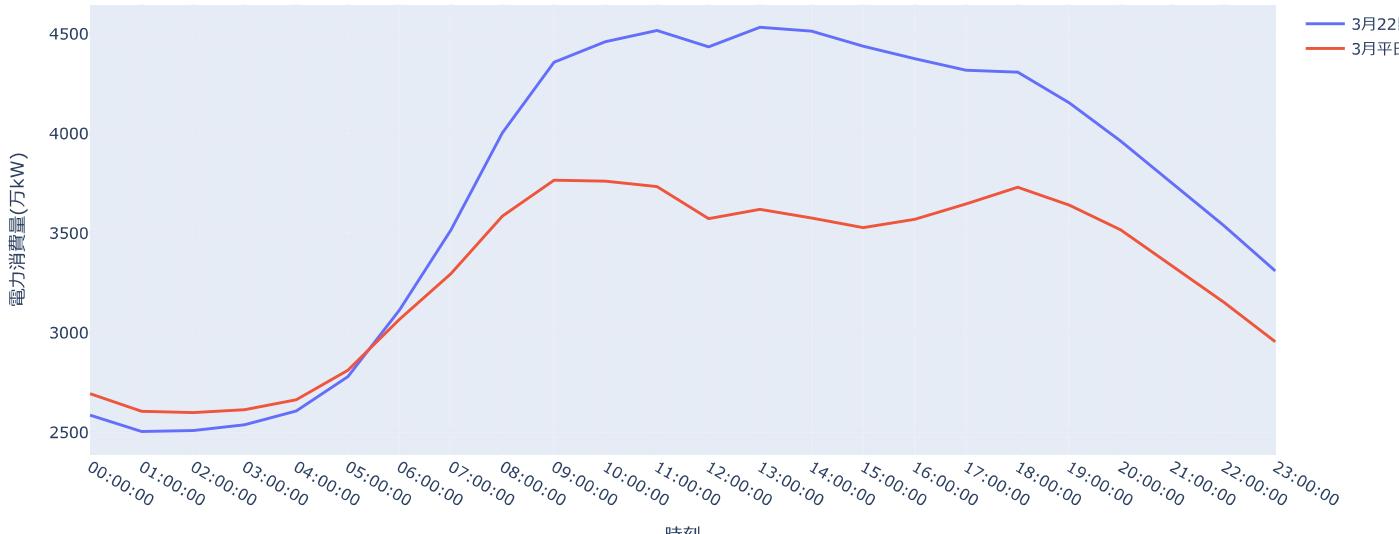
# 3月平日平均のデータをプロット
fig.add_trace(
    go.Scatter(
        x=march_weekday_avg["time"],
        y=march_weekday_avg["10MW"],
        mode="lines",
        name="3月平日平均",
    )
)

# x軸を時刻として設定
fig.update_xaxes(title_text="時刻", tickformat="%H:%M")

# グラフタイトルと表示
fig.update_layout(
    title="2022年3月22日の電力消費量と3月平日平均の比較",
    xaxis_title="時刻",
    yaxis_title="電力消費量(万kW)",
)
```

```
fig.show()
```

2022年3月22日の電力消費量と3月平日平均の比較



- 3月の平日平均と比べてかなり電力消費量が多いことが分かった
- 原因の一つとして、季節外れの低温が考えられるため気温を抽出する

```
In [43]: # 気温データ (temperature_2022) から3月のデータを抽出
march_temperatures = merged_data[
    (merged_data.index.year == 2022) & (merged_data.index.month == 3)
]
march_temperatures["date"] = march_temperatures.index.date
march_temperatures
```

```
Out[43]:
```

	10MW	temperature	year	month	hour	week	week_jp	is_holiday	date
	DATETIME								
2022-03-01 00:00:00	2876	8.8	2022	3	0	1	火	0	2022-03-01
2022-03-01 01:00:00	2788	8.2	2022	3	1	1	火	0	2022-03-01
2022-03-01 02:00:00	2778	7.5	2022	3	2	1	火	0	2022-03-01
2022-03-01 03:00:00	2783	7.0	2022	3	3	1	火	0	2022-03-01
2022-03-01 04:00:00	2844	6.0	2022	3	4	1	火	0	2022-03-01
...
2022-03-31 19:00:00	3206	13.9	2022	3	19	3	木	0	2022-03-31
2022-03-31 20:00:00	3112	13.0	2022	3	20	3	木	0	2022-03-31
2022-03-31 21:00:00	2980	11.4	2022	3	21	3	木	0	2022-03-31
2022-03-31 22:00:00	2857	10.6	2022	3	22	3	木	0	2022-03-31
2022-03-31 23:00:00	2714	9.6	2022	3	23	3	木	0	2022-03-31

744 rows × 9 columns

```
In [44]: # 日ごとの平均気温を計算
march_avg_temperatures = pd.DataFrame(march_temperatures.groupby("date")[[ "10MW", "temperature"]].mean())
march_avg_temperatures.index = pd.to_datetime(march_avg_temperatures.index)
march_avg_temperatures
```

Out[44]:

10MW temperature

date

2022-03-01	3447.125000	10.958333
2022-03-02	3337.208333	11.279167
2022-03-03	3361.125000	9.500000
2022-03-04	3492.291667	8.050000
2022-03-05	2981.625000	11.070833
2022-03-06	2968.083333	8.254167
2022-03-07	3523.791667	8.241667
2022-03-08	3742.041667	6.304167
2022-03-09	3512.750000	7.941667
2022-03-10	3479.041667	8.908333
2022-03-11	3275.791667	11.775000
2022-03-12	2759.375000	14.583333
2022-03-13	2629.625000	14.491667
2022-03-14	2893.375000	16.812500
2022-03-15	3014.291667	12.579167
2022-03-16	3031.750000	14.070833
2022-03-17	3002.916667	14.112500
2022-03-18	3605.083333	5.404167
2022-03-19	2968.958333	9.950000
2022-03-20	2788.250000	9.808333
2022-03-21	2935.666667	9.516667
2022-03-22	3714.375000	4.033333
2022-03-23	3678.041667	5.129167
2022-03-24	3427.208333	8.041667
2022-03-25	3211.833333	11.975000
2022-03-26	2822.208333	16.012500
2022-03-27	2528.125000	16.429167
2022-03-28	2936.000000	13.550000
2022-03-29	3263.666667	10.200000
2022-03-30	2996.000000	14.283333
2022-03-31	2885.291667	15.300000

In [45]: `march_avg_temperatures.corr()`

Out[45]:

10MW temperature

10MW	1.000000	-0.821701
temperature	-0.821701	1.000000

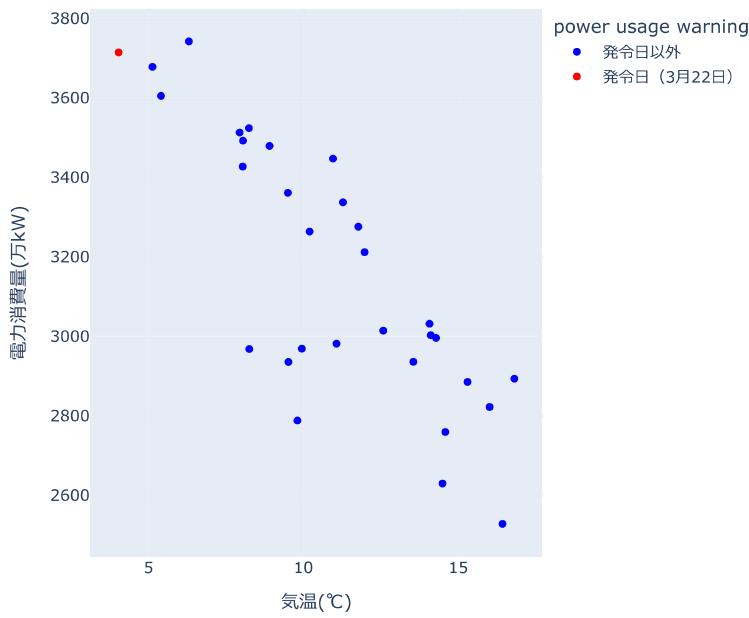
In [46]: `# 3月22日のデータに色を付けるための列を追加`

```

# Index オブジェクトには .apply() メソッドが直接使えないで pd.Series に変換して処理する
march_avg_temperatures['power_usage_warning'] = march_avg_temperatures.index.to_series().apply(
    lambda x: "発令日（3月22日）" if x.strftime('%Y-%m-%d') == '2022-03-22' else "発令日以外"
)
fig = px.scatter(
    march_avg_temperatures,
    x="temperature",
    y="10MW",
    title="気温と電力消費量の関係（2022年3月）",
    labels={"temperature": "気温(℃)", "10MW": "電力消費量(万kW)" },
    color="power_usage_warning", # 色を3月22日かどうかで分ける
    color_discrete_map={"発令日（3月22日）": "red", "発令日以外": "blue"} # 3月22日は赤、それ以外は青
)
# グラフを正方形に設定
fig.update_layout(
    width=600, # 正方形の幅
    height=600, # 正方形の高さ
)
fig.show()

```

気温と電力消費量の関係 (2022年3月)



- 3月22日は3月の中でも特に気温が低くなっていたことにより、電力需要が大幅に増加したことが原因であると考えられる。
- 電力使用量と気温における散布図と相関係数より、明確に負の相関が見受けられ、気温を説明変数にすれば適切に予測ができると考えられる。
- よって、今回は外れ値とも見なさない。

仮説5. コロナ禍の影響

- 仮説：緊急事態宣言中は電力消費量が減少する。
- 手法：緊急事態宣言下と平常時の電力消費量の違いを可視化する

補足：東京都の緊急事態宣言の発令期間

- 第1回緊急事態宣言
 - 期間: 2020年4月7日～2020年5月25日
 - 内容: 初めての緊急事態宣言が発令され、外出自粛や一部の業種の営業停止が要請された。
- 第2回緊急事態宣言
 - 期間: 2021年1月8日～2021年3月21日
 - 内容: 感染再拡大に伴い、再び緊急事態宣言が発令され、飲食店の営業時間短縮や不要不急の外出自粛が要請された。
- 第3回緊急事態宣言
 - 期間: 2021年4月25日～2021年6月20日
 - 内容: ゴールデンウィーク前に感染が再拡大したため、再度の緊急事態宣言が発令された。
- 第4回緊急事態宣言
 - 期間: 2021年7月12日～2021年9月30日
 - 内容: 東京オリンピック開催中も緊急事態宣言下にあり、飲食店への休業要請や外出自粛が継続された。
- 参考: [令和4年版 犯罪白書 第7編/第2章/3](#)

```
In [47]: # 緊急事態宣言の期間と対応するフラグをリストで定義
emergency_periods = [
    {"start_date": "2020-04-07", "end_date": "2020-05-25", "flag": 1},
    {"start_date": "2021-01-08", "end_date": "2021-03-21", "flag": 2},
    {"start_date": "2021-04-25", "end_date": "2021-06-20", "flag": 3},
    {"start_date": "2021-07-12", "end_date": "2021-09-30", "flag": 4},
]
emergency_periods
```

```
Out[47]: [{"start_date": '2020-04-07', 'end_date': '2020-05-25', 'flag': 1},
{'start_date': '2021-01-08', 'end_date': '2021-03-21', 'flag': 2},
{'start_date': '2021-04-25', 'end_date': '2021-06-20', 'flag': 3},
{'start_date': '2021-07-12', 'end_date': '2021-09-30', 'flag': 4}]
```

```
In [48]: # corona_emergency データフレームを作成
corona_emergency = pd.DataFrame(emergency_periods)
corona_emergency
```

	start_date	end_date	flag
0	2020-04-07	2020-05-25	1
1	2021-01-08	2021-03-21	2
2	2021-04-25	2021-06-20	3
3	2021-07-12	2021-09-30	4

```
In [49]: # 新しいフラグ列を作成し、初期値は0  
merged_data["emergency_flag"] = 0  
merged_data.head()
```

```
Out[49]:
```

DATETIME	10MW	temperature	year	month	hour	week	week_jp	is_holiday	emergency_flag
2016-04-01 00:00:00	2555	14.5	2016	4	0	4	金	0	0
2016-04-01 01:00:00	2433	13.3	2016	4	1	4	金	0	0
2016-04-01 02:00:00	2393	13.8	2016	4	2	4	金	0	0
2016-04-01 03:00:00	2375	13.0	2016	4	3	4	金	0	0
2016-04-01 04:00:00	2390	12.2	2016	4	4	4	金	0	0

```
In [50]: # 各緊急事態宣言の期間に対して、該当する日付にフラグを設定
```

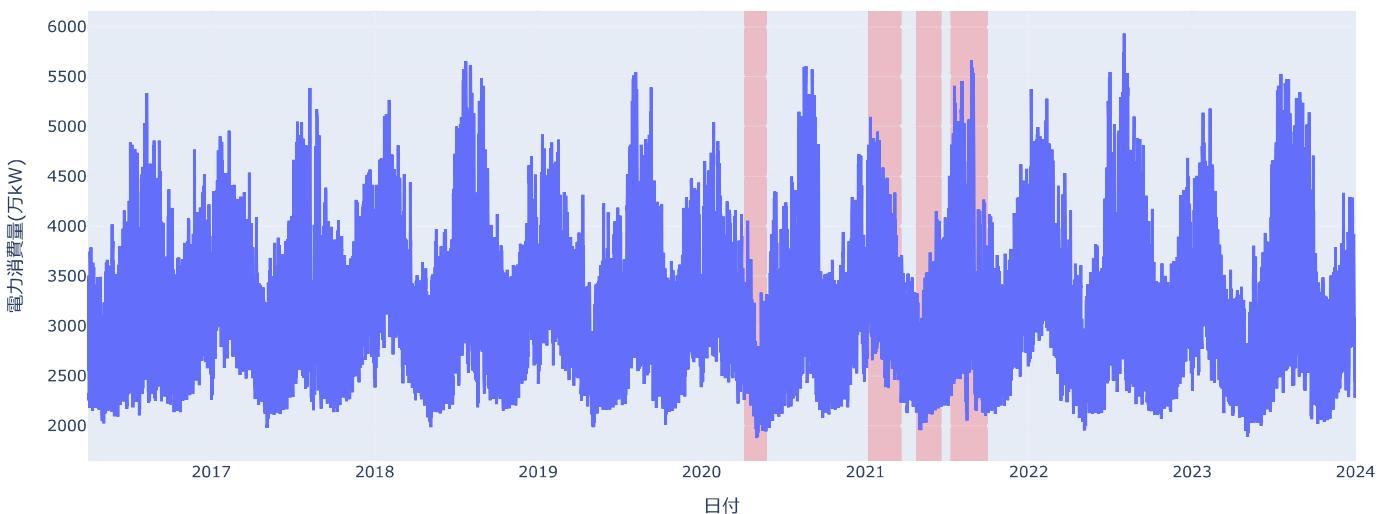
```
for period in emergency_periods:  
    start_date = pd.to_datetime(period["start_date"])  
    end_date = pd.to_datetime(period["end_date"])  
    flag_value = period["flag"]  
  
    # 該当する期間のデータにフラグを設定  
    merged_data.loc[  
        (merged_data.index >= start_date) & (merged_data.index <= end_date),  
        "emergency_flag",  
    ] = flag_value
```

```
In [51]: # 緊急事態宣言期間の帯を描画する準備  
fig = go.Figure()
```

```
# 電力消費量の時系列データを追加  
fig.add_trace(  
    go.Scatter(  
        x=merged_data.index, y=merged_data["10MW"], mode="lines", name="電力消費量"  
    )
)  
  
# 緊急事態宣言期間に帯を追加  
for _, row in corona_emergency.iterrows():  
    fig.add_vrect(  
        x0=row["start_date"],  
        x1=row["end_date"],  
        fillcolor="red",  
        opacity=0.2,  
        layer="below",  
        line_width=0,  
    )
  
# グラフのレイアウト設定  
fig.update_layout(  
    title="電力消費量の推移と緊急事態宣言期間",  
    xaxis_title="日付",  
    yaxis_title="電力消費量(万kW)",  

)
fig.show()
```

電力消費量の推移と緊急事態宣言期間



- 上図はコロナの緊急事態宣言期間を赤帯で示している
- 本グラフを見る限り、緊急事態宣言下に大きな落ち込みは見られない
- よって、コロナ禍前と緊急事態宣言下、各緊急事態宣言の間の期間、コロナ禍後に区切って箱ひげ図でどれくらい変動があるかを調べる

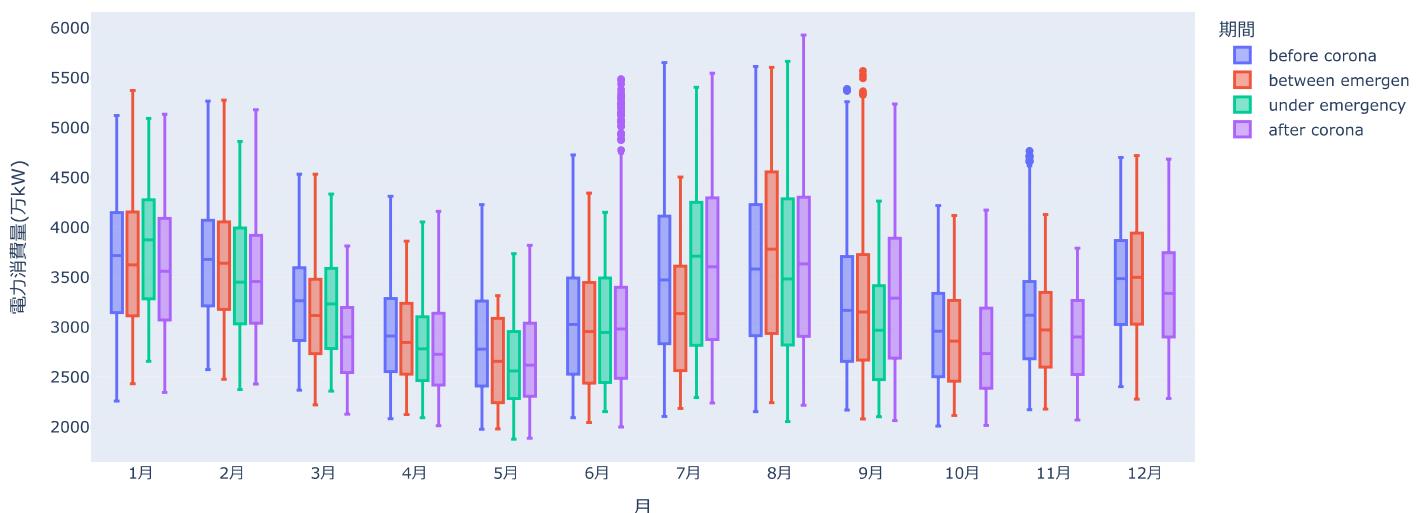
```
In [52]: # 箱ひげ図で比較するための準備
merged_data["period"] = "between emergencies"
# 緊急事態宣言期間に対してフラグを設定
for period in emergency_periods:
    merged_data.loc[
        (merged_data.index >= period["start_date"])
        & (merged_data.index <= period["end_date"]),
        "period",
    ] = "under emergency"
# コロナ前の期間を 'before corona' とする
merged_data.loc[
    (merged_data.index >= "2016-04-01") & (merged_data.index <= "2019-12-31"), "period"
] = "before corona"
# コロナ後の期間を 'after corona' とする
merged_data.loc[
    (merged_data.index >= "2022-04-01") & (merged_data.index <= "2023-12-31"), "period"
] = "after corona"
merged_data.head()
```

Out[52]:

	10MW	temperature	year	month	hour	week	week_jp	is_holiday	emergency_flag	period
DATETIME										
2016-04-01 00:00:00	2555	14.5	2016	4	0	4	金	0	0	before corona
2016-04-01 01:00:00	2433	13.3	2016	4	1	4	金	0	0	before corona
2016-04-01 02:00:00	2393	13.8	2016	4	2	4	金	0	0	before corona
2016-04-01 03:00:00	2375	13.0	2016	4	3	4	金	0	0	before corona
2016-04-01 04:00:00	2390	12.2	2016	4	4	4	金	0	0	before corona

```
In [53]: # コロナ前、コロナ後、および緊急事態宣言期間のデータを用いてボックスプロットを作成
fig = px.box(
    merged_data,
    x="month",
    y="10MW",
    color="period",
    title="コロナ前、緊急事態宣言下、コロナ後の対応する月ごとの電力消費量の比較",
    labels={"month": "月", "10MW": "電力消費量(万kW)", "period": "期間"},
)
# x軸の設定:1月から12月までを表示
fig.update_xaxes(
    tickmode="array",
    tickvals=list(range(1, 13)),
    ticktext=[
        "1月", "2月",
        "3月", "4月",
        "5月", "6月",
        "7月", "8月",
        "9月", "10月",
        "11月", "12月",
    ],
)
fig.show()
```

コロナ前、緊急事態宣言下、コロナ後の対応する月ごとの電力消費量の比較



- 10-12月は緊急事態宣言が発令された期間がなかったため、その期間は緑色の箱ひげ図がないグラフになっている。
- 緊急事態宣言下および各発令の間の期間で外れ値になっているような値は見受けられない。

```
In [54]: summary_data = merged_data.groupby(['period', 'month'])['10MW'].mean().astype(int).reset_index()
pivot_table = summary_data.pivot(index='month', columns='period', values='10MW')
pivot_table
```

```
Out[54]: period after corona before corona between emergencies under emergency
```

month	1	3589.0	3661.0	3654.0	3817.0
2	3495.0	3661.0	3641.0	3498.0	
3	2880.0	3257.0	3130.0	3214.0	
4	2779.0	2923.0	2875.0	2800.0	
5	2676.0	2835.0	2664.0	2626.0	
6	3046.0	3045.0	2987.0	2994.0	
7	3649.0	3525.0	3143.0	3646.0	
8	3684.0	3620.0	3797.0	3604.0	
9	3342.0	3227.0	3280.0	2991.0	
10	2780.0	2952.0	2883.0	Nan	
11	2900.0	3102.0	2983.0	Nan	
12	3340.0	3461.0	3489.0	Nan	

- 上表は各期間における月別の平均電力使用量を表している。
- これだけだと期間ごとに優位な差があるかどうかが分からないので二元配置分散分析（ANOVA）を実施したい。
 - 要因1:「期間 (before corona, after corona, under emergency, between emergencies)」
 - 要因2:「月 (1月～12月)」
- また、3つ以上のサンプル間の、どの組み合わせに有意差があるかを見つけるために使用されるテューキーのHSD（範囲）検定も実施したい。

```
In [55]: from statsmodels.formula.api import ols
import statsmodels.api as sm
from statsmodels.stats.multicomp import pairwise_tukeyhsd
# カラム名を変更
anova_data = merged_data[['10MW', 'period', 'month']].dropna().rename(columns={'10MW': 'power_usage'})
# "under_emergency" フラグを作成
anova_data["under_emergency"] = (anova_data["period"] == "under emergency").astype(int)

# 線形モデル
model = ols('power_usage ~ C(under_emergency) + C(month)', data=anova_data).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)

# Tukey's HSD 検定
tukey = pairwise_tukeyhsd(endog=anova_data["power_usage"], groups=anova_data["period"], alpha=0.05)
print(tukey)
```

	sum_sq	df	F	PR(>F)
C(under_emergency)	1.907572e+07	1.0	54.893569	1.287063e-13
C(month)	6.795818e+09	11.0	1777.827231	0.000000e+00
Residual	2.360628e+10	67931.0	Nan	Nan

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
after corona	before corona	95.0943	0.0	78.3173	111.8712	True
after corona	between emergencies	99.4616	0.0	79.256	119.6672	True
after corona	under emergency	6.2355	0.9262	-19.6605	32.1315	False
before corona	between emergencies	4.3673	0.9185	-13.1239	21.8586	False
before corona	under emergency	-88.8588	0.0	-112.6974	-65.0202	True
between emergencies	under emergency	-93.2261	0.0	-119.5905	-66.8618	True

1. ANOVAの結果

- C(under_emergency) の p 値が極めて小さい (1.28e-13) ため、緊急事態宣言期間 (under emergency) が電力消費量に有意な影響を与えている。
- 月ごとの効果 (C(month)) も非常に強く、p 値は 0.000 である。

2. Tukey's HSDの結果

- after corona vs. before corona
 - 差分が 95.0943 で、有意差あり (p 値 = 0.0)。
 - コロナ後の電力消費量はコロナ前よりわずかに高い。
- after corona vs. under emergency
 - 差分が 6.2355 で、有意差なし (p 値 = 0.9262)。
 - 緊急事態宣言期間中とコロナ後の電力消費量に大きな差は見られない。
- before corona vs. under emergency
 - 差分が -88.8588 で、有意差あり (p 値 = 0.0)。
 - 緊急事態宣言期間中の電力消費量はコロナ前より低い。
- between emergencies vs. under emergency
 - 差分が -93.2261 で、有意差あり (p 値 = 0.0)。
 - 緊急事態宣言期間中は、緊急事態宣言間の通常期間よりも消費量が低い。

よって、緊急事態宣言期間中の電力消費量は、コロナ前や緊急事態宣言間の通常期間 (between emergencies) よりも明確に低い。一方で、コロナ後の期間と比較すると有意な差は見られない。

結論 緊急事態宣言期間をモデルに組み込むことは、予測性能の向上には寄与しない可能性が高いと判断し、説明変数としては使用しないこととした。

仮説検証のまとめ

1. 季節性の影響

- 電気の使用量は明確な季節性を持ち、夏期と冬期に多く、春期と秋期には少ない傾向が見られた。
- これは夏の冷房や冬の暖房需要に起因していると考えられる。

2. 休日と平日の違い

- 土日・祝日は平日と比べて電力消費量が少なく、休日か平日かで消費量に顕著な違いが確認できた。
- 産業活動が減少し、家庭消費が主体となるためと推測される。

3. 時間帯による変動

- 一日の電力消費量の推移は、年間を通して安定しており、日中にピークを迎え、夜間には減少する傾向が確認された。
- 日中の産業活動と夜間の家庭需要の影響と考えられる。

4. 電力逼迫警報時の異常値

- 2022年3月22日、東京エリアに電力逼迫警報が発令された日は、低温が原因で電力消費量が大幅に増加していた。このような異常値は、気温と電力消費量の関係性が顕著に表れた例と考える。

5. コロナ禍の影響

- コロナ禍における緊急事態宣言期間 (under emergency) は、電力消費量に有意な影響を与えていないことが確認された。
- 緊急事態宣言期間を含む「コロナ前 (before corona)」「緊急事態宣言下 (under emergency)」「緊急事態宣言間 (between emergency)」「コロナ後 (after corona)」の期間比較では、緊急事態宣言下が他の期間と大きく異なる傾向は見られなかった。モデル構築時に説明変数として組み込む必要性は低いと判断した。
- 「ポートフォリオ 電力消費予測 モデル化」に続く
- また、本ファイルで作ったデータは以下のように"data_csv"として書き出してモデル化の際に使うことしたい。

```
In [56]: # CSVファイルを作成  
#merged_data.to_csv('data.csv', index=True, encoding='utf-8')
```

以上