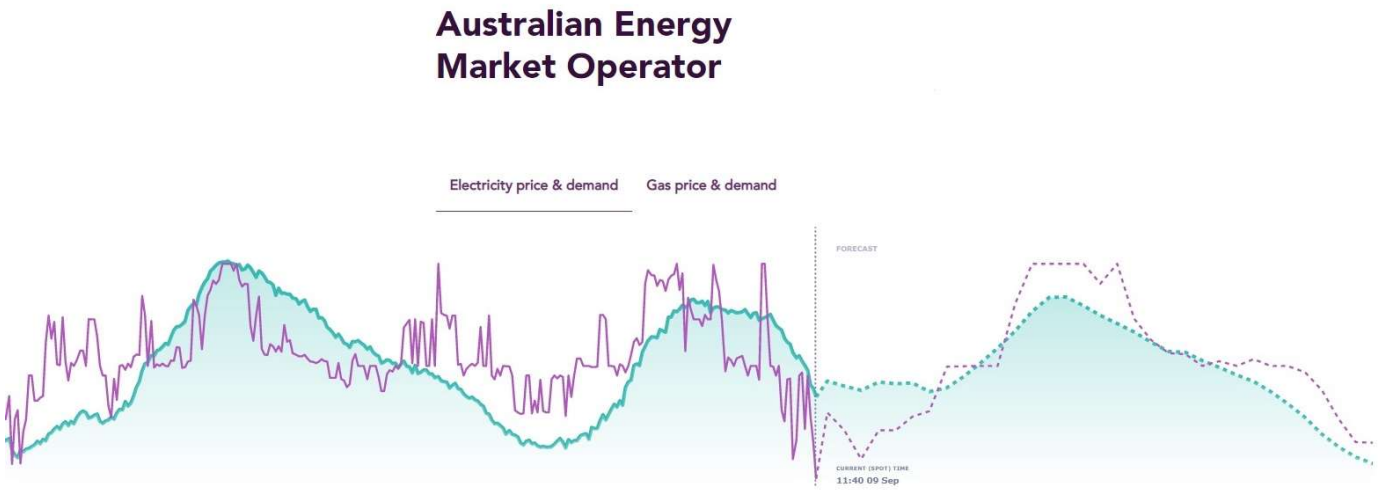
Max_Daily_Energy_Predictions_Project

Max Daily Energy Prediction using Regression Analysis

Author - Udhai P Singh



Project Overview

This project analyzes the electricity consumption of Melbourne from 01.01.2021 to 31.08.2021. It is often observed that electricity consumption has strong relation with weather, especially temperature and time of the year/season. We will analyze the relationship of electricity consumption in Melbourne based on various weather features like temperature, wind, month and so on. The goal here is to predict the electricity consumption of the city based on weather forecast.

Our stakeholders, the electricity providers, can use this model to plan for the expected demand and hence make arrangements as participants of National Electricity Market(NEM) to lock in the expected demand at the right price.

Data:

The following two datasets are used:

1. price_demand_data.csv: This excel contains price and demand for each 30 minutes period from 01.01.2021 to 31.08.2021, for Victoria, Australia. RRP column has been removed from the data. For more details, please refer to the data source: https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data (https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data)
2. weather_data.csv: contains various weather features like temperature, wind pressure, wind direction and so forth, for each day from 01.01.2021 to 31.08.2021. Note, that these readings are on daily interval ie. one readign per day.

Busines Problem

The retail electricity provider participates in NEM in the spot market to procure electricity to sell it to consumers. NEM participants need to manage the financial risks associated with the significant spot price volatility that occurs during trading periods. Providers hedge their risk by locking in a firm price for electricity that will be produced or consumed at a given time in the future. These arrangements are generally in the form of derivatives, and include swaps or hedges, options and futures contracts.

In order to enter various purchasing arrangements at the right price, the demand forecast becomes critical especially the maximum demands for which the retails need to be prepared, otherwise purchasing units from the spot market at last minute to meet unexpected peak demand is exorbitantly expensive as compared to long term derivatives.

To solve this problem for our stakeholders, we are trying to predict the daily maximum electricity demand based on weather forecast for Melbourne.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from sklearn.preprocessing import LabelEncoder
from IPython import display

%matplotlib inline
```

In [2]:

```
weather_df = pd.read_csv("weather_data.csv",  parse_dates = ['Date'],sep = ',', dayfirs
```

In [3]:

```
weather_df.head(5)
```

Out[3]:

	Date	Minimum temperature (°C)	Maximum temperature (°C)	Rainfall (mm)	Evaporation (mm)	Sunshine (hours)	Direction of maximum wind gust	Speed of maximum wind gust (km/h)	T max win
0	2021-01-01	15.6	29.9	0.0	2.8	9.3	NNE	31.0	
1	2021-01-02	18.4	29.0	0.0	9.4	1.3	NNW	30.0	
2	2021-01-03	17.0	26.2	12.6	4.8	7.1	WSW	33.0	
3	2021-01-04	16.0	18.6	2.6	3.8	0.0	SSE	41.0	
4	2021-01-05	15.9	19.1	11.2	1.0	0.0	SSE	35.0	

5 rows × 21 columns

In [4]:

```
1 weather_df.describe()
```

Out[4]:

	Minimum temperature (°C)	Maximum temperature (°C)	Rainfall (mm)	Evaporation (mm)	Sunshine (hours)	Speed of maximum wind gust (km/h)	9am Temperature (°C)
count	242.000000	242.000000	241.000000	243.000000	243.000000	240.000000	242.000000
mean	11.050826	19.445868	1.576763	3.902469	5.349383	34.412500	13.720661
std	3.870242	5.354085	4.498754	2.702141	3.604902	10.909319	4.306618
min	1.700000	10.600000	0.000000	0.000000	0.000000	15.000000	3.000000
25%	8.100000	15.500000	0.000000	1.900000	2.150000	28.000000	10.925000
50%	10.900000	18.300000	0.000000	3.200000	4.900000	33.000000	13.400000
75%	13.800000	21.800000	0.600000	5.600000	8.350000	41.000000	16.400000
max	22.200000	39.200000	43.200000	13.800000	13.100000	67.000000	30.900000

In [5]:

```
1 weather_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243 entries, 0 to 242
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   Date                             243 non-null    datetime64[ns]
 1   Minimum temperature (°C)         242 non-null    float64
 2   Maximum temperature (°C)         242 non-null    float64
 3   Rainfall (mm)                    241 non-null    float64
 4   Evaporation (mm)                 243 non-null    float64
 5   Sunshine (hours)                 243 non-null    float64
 6   Direction of maximum wind gust   240 non-null    object
 7   Speed of maximum wind gust (km/h)  240 non-null    float64
 8   Time of maximum wind gust        240 non-null    object
 9   9am Temperature (°C)             242 non-null    float64
 10  9am relative humidity (%)        242 non-null    float64
 11  9am cloud amount (oktas)         243 non-null    int64
 12  9am wind direction               242 non-null    object
 13  9am wind speed (km/h)            242 non-null    object
 14  9am MSL pressure (hPa)           241 non-null    float64
 15  3pm Temperature (°C)             243 non-null    float64
 16  3pm relative humidity (%)        243 non-null    int64
 17  3pm cloud amount (oktas)         242 non-null    float64
 18  3pm wind direction               243 non-null    object
 19  3pm wind speed (km/h)            243 non-null    object
 20  3pm MSL pressure (hPa)           242 non-null    float64
dtypes: datetime64[ns](1), float64(12), int64(2), object(6)
memory usage: 40.0+ KB
```

In [6]:

```
price_demand_df = pd.read_csv("price_demand_data.csv", parse_dates = ['SETTLEMENTDATE']
```

In [7]:

```
price_demand_df.head(5)
```

Out[7]:

	REGION	SETTLEMENTDATE	TOTALDEMAND	PRICECATEGORY
0	VIC1	2021-01-01 00:30:00	4179.21	LOW
1	VIC1	2021-01-01 01:00:00	4047.76	LOW
2	VIC1	2021-01-01 01:30:00	3934.70	LOW
3	VIC1	2021-01-01 02:00:00	3766.45	LOW
4	VIC1	2021-01-01 02:30:00	3590.37	LOW

In [8]:

```
# We will apply label encoder on the dependant variable PriceCategory

label_encoder = LabelEncoder()
price_demand_df['PRICECATEGORY'] = label_encoder.fit_transform(price_demand_df['PRICECA
```

In [9]:

```
price_demand_df.describe()
```

Out[9]:

	TOTALDEMAND	PRICECATEGORY
count	11664.000000	11664.000000
mean	4925.798454	2.151920
std	876.407490	0.627529
min	2708.530000	0.000000
25%	4255.500000	2.000000
50%	4803.755000	2.000000
75%	5477.337500	3.000000
max	8196.830000	3.000000

In [10]:

```
price_demand_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11664 entries, 0 to 11663
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   REGION          11664 non-null  object
 1   SETTLEMENTDATE  11664 non-null  datetime64[ns]
 2   TOTALDEMAND     11664 non-null  float64
 3   PRICECATEGORY   11664 non-null  int32
dtypes: datetime64[ns](1), float64(1), int32(1), object(1)
memory usage: 319.1+ KB
```

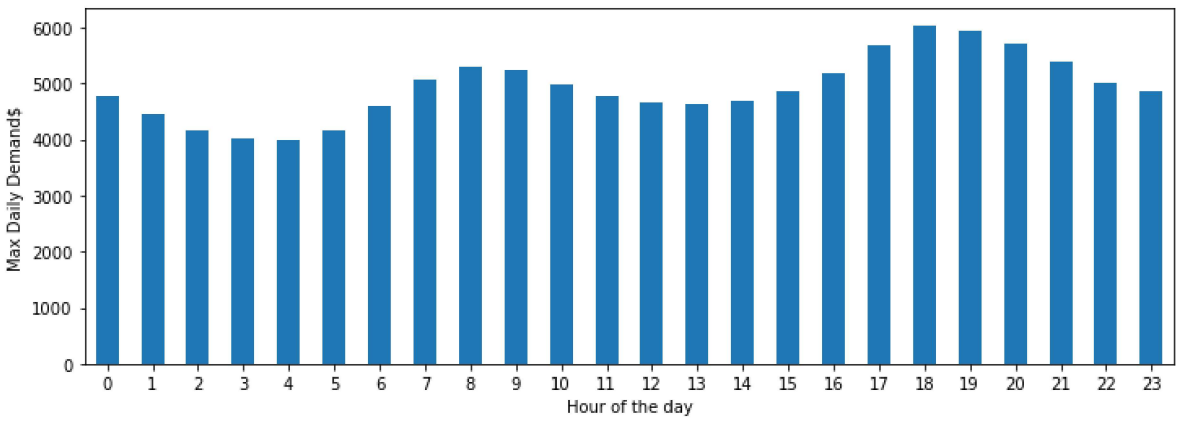
In [11]:

```
price_demand_df['Date'] = price_demand_df.SETTLEMENTDATE
price_demand_df.Date = pd.to_datetime(price_demand_df['Date']).dt.date
price_demand_df.TOTALDEMAND = price_demand_df.TOTALDEMAND.astype(int)
gp1 = price_demand_df.groupby('Date').TOTALDEMAND.max().reset_index()
gp2 = price_demand_df.groupby('Date').PRICECATEGORY.max().reset_index()
```

Hourly Trend

In [12]:

```
fig, axs = plt.subplots(figsize=(12, 4))

price_demand_df.groupby(price_demand_df["SETTLEMENTDATE"].dt.hour)["TOTALDEMAND"].mean(
    kind='bar', rot=0, ax=axs
)

plt.xlabel("Hour of the day");  # custom x label using Matplotlib

plt.ylabel("Max Daily Demand$");
```



As expected, the hour of the day matters even on a eight month span. Electricity consumption is maximum at 6pm in the evening. Whereas the mean hourly electricity consumption dips to a minimum at 3 am. So this should inform at what time most generators come on and off the grid. This should inform the hourly forecast of the retail distributors.

In [13]:

```
gp1.drop([243], inplace=True)
gp2.drop([243], inplace=True)
gp3 = gp1.join(gp2.set_index('Date'), on='Date')
```

In [14]:

```
weather_df['9am wind direction'] = weather_df['9am wind direction'].replace(" ","missir
weather_df['3pm wind direction'] = weather_df['3pm wind direction'].replace(" ","missir

weather_df['9am wind speed (km/h)'] = weather_df['9am wind speed (km/h)'].replace("Calr
weather_df['3pm wind speed (km/h)'] = weather_df['3pm wind speed (km/h)'].replace("Calr

```

In [15]:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='constant', fill_value='missing')

directions_df = pd.DataFrame(imputer.fit_transform(weather_df[['Direction of maximum wi
 '9am wind direction']]))

weather_df[['Direction of maximum wind gust ',
 '9am wind direction']] = directions_df[[0,1]]
```

These are readings of the 16 point compass. Hence, there is an order and relation of these categoreies with each other. We will proceed ahead to encode the 16 points as ordinal variable and thus aim to get the relationship of particular direction and demand.

In [16]:

```
directions = ['Direction of maximum wind gust ', '9am wind direction', '3pm wind direct

sixteen_pts = ['N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE', 'S', 'SSW', 'SW', 'WSW

from sklearn.preprocessing import OrdinalEncoder

oe = OrdinalEncoder(categories=[['N', 'NNE', 'NE', 'ENE', 'E', 'ESE', 'SE', 'SSE', 'S',
pd.DataFrame(oe.fit_transform(weather_df[['Direction of maximum wind gust ', '9am wind

weather_df[['Direction of maximum wind gust ', '9am wind direction', '3pm wind directio

weather_df.head(5)

```

Out[16]:

	Date	Minimum temperature (°C)	Maximum temperature (°C)	Rainfall (mm)	Evaporation (mm)	Sunshine (hours)	Direction of maximum wind gust	Speed of maximum wind gust (km/h)	T max win
0	2021-01-01	15.6	29.9	0.0	2.8	9.3	1.0	31.0	
1	2021-01-02	18.4	29.0	0.0	9.4	1.3	15.0	30.0	
2	2021-01-03	17.0	26.2	12.6	4.8	7.1	11.0	33.0	
3	2021-01-04	16.0	18.6	2.6	3.8	0.0	7.0	41.0	
4	2021-01-05	15.9	19.1	11.2	1.0	0.0	7.0	35.0	

5 rows × 21 columns

In [17]:

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
impute_it = IterativeImputer()
weather_df[['Minimum temperature (°C)','Maximum temperature (°C)','Rainfall (mm)',
                                    'Speed of maximum wind gust (km/h)',
                                   '9am Temperature (°C)','9am relative humidity (%)',
                                    '3pm cloud amount (oktas)', '3pm wind speed (km/h)
                                    'Speed of maximum wind gust (km/h)',
                                   '9am Temperature (°C)', '9am relative humidity (%)',
                                    '3pm cloud amount (oktas)', '3pm wind speed (km/h)

```

In [18]:

```
## Dropping Time of maximum wind gust, as the day to day variability is too high and se
## Infact, in order to capture any impact of time, we will create separate month, weeke
## Subsequency after merging the two datasets, we should check the correlations of thes
## dependant variables

weather_df.drop(columns=['Time of maximum wind gust'], axis=1, inplace=True)
weather_df['month'] = weather_df.Date.dt.month
weather_df['day_of_week'] = weather_df.Date.dt.day_of_week

```

Merging both the datasets

In [19]:

```
final_df = pd.concat([weather_df, gp3.iloc[:,1:3]], axis=1)
final_df.head(5)
```

Out[19]:

	Date	Minimum temperature (°C)	Maximum temperature (°C)	Rainfall (mm)	Evaporation (mm)	Sunshine (hours)	Direction of maximum wind gust	Speed of maximum wind gust (km/h)	Tem
0	2021-01-01	15.6	29.9	0.0	2.8	9.3	1.0	31.0	
1	2021-01-02	18.4	29.0	0.0	9.4	1.3	15.0	30.0	
2	2021-01-03	17.0	26.2	12.6	4.8	7.1	11.0	33.0	
3	2021-01-04	16.0	18.6	2.6	3.8	0.0	7.0	41.0	
4	2021-01-05	15.9	19.1	11.2	1.0	0.0	7.0	35.0	

5 rows × 24 columns

In [20]:

```
X = final_df.iloc[:,1:22]
y1 = final_df.iloc[:,22:23]
y2 = final_df.iloc[:,23:24]
```

In [21]:

```
corr= final_df.corr()
corr.sort_values(["TOTALDEMAND"], ascending = False, inplace = True)
print("Correlation with TOTALDEMAND showing the most important features in decreasing c
print(corr.TOTALDEMAND)
```

```
Correlation with TOTALDEMAND showing the most important features in decreasi
ng order:
TOTALDEMAND                          1.000000
month                                0.574329
PRICECATEGORY                        0.507024
9am wind speed (km/h)                0.127071
9am relative humidity (%)            0.104367
3pm cloud amount (oktas)             0.068739
Speed of maximum wind gust (km/h)    0.068069
3pm relative humidity (%)            0.064331
9am MSL pressure (hPa)               0.055608
3pm MSL pressure (hPa)               0.000427
3pm wind speed (km/h)               -0.035397
Rainfall (mm)                       -0.073171
Direction of maximum wind gust      -0.078329
Sunshine (hours)                    -0.139570
3pm wind direction                  -0.165738
9am cloud amount (oktas)            -0.167404
day_of_week                         -0.240428
Evaporation (mm)                    -0.264013
Maximum temperature (°C)            -0.290422
9am wind direction                  -0.298344
3pm Temperature (°C)                -0.325236
9am Temperature (°C)                -0.391110
Minimum temperature (°C)            -0.488118
Name: TOTALDEMAND, dtype: float64
```

Groupby Exploration

In [22]:

```
data = final_df.copy()
monthly_df = data.groupby(['month','day_of_week']).TOTALDEMAND.agg(["max", "min", "cour
monthly_df.head(5)
```

Out[22]:

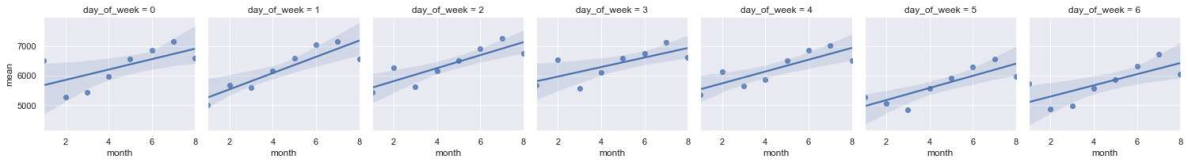
	month	day_of_week	max	min	count	median	mean
0	1	0	8196	4764	4	6538.5	6509.25
1	1	1	5349	4800	4	4925.5	5000.00
2	1	2	6522	4691	4	5256.0	5431.25
3	1	3	7046	4994	4	5327.5	5673.75
4	1	4	5954	4742	5	5365.0	5363.40

In [23]:

```
sns.set(rc={"figure.figsize": (18,18)})
g = sns.FacetGrid(monthly_df, col="day_of_week")
g.map(sns.regplot, "month", "mean")
```

Out[23]:

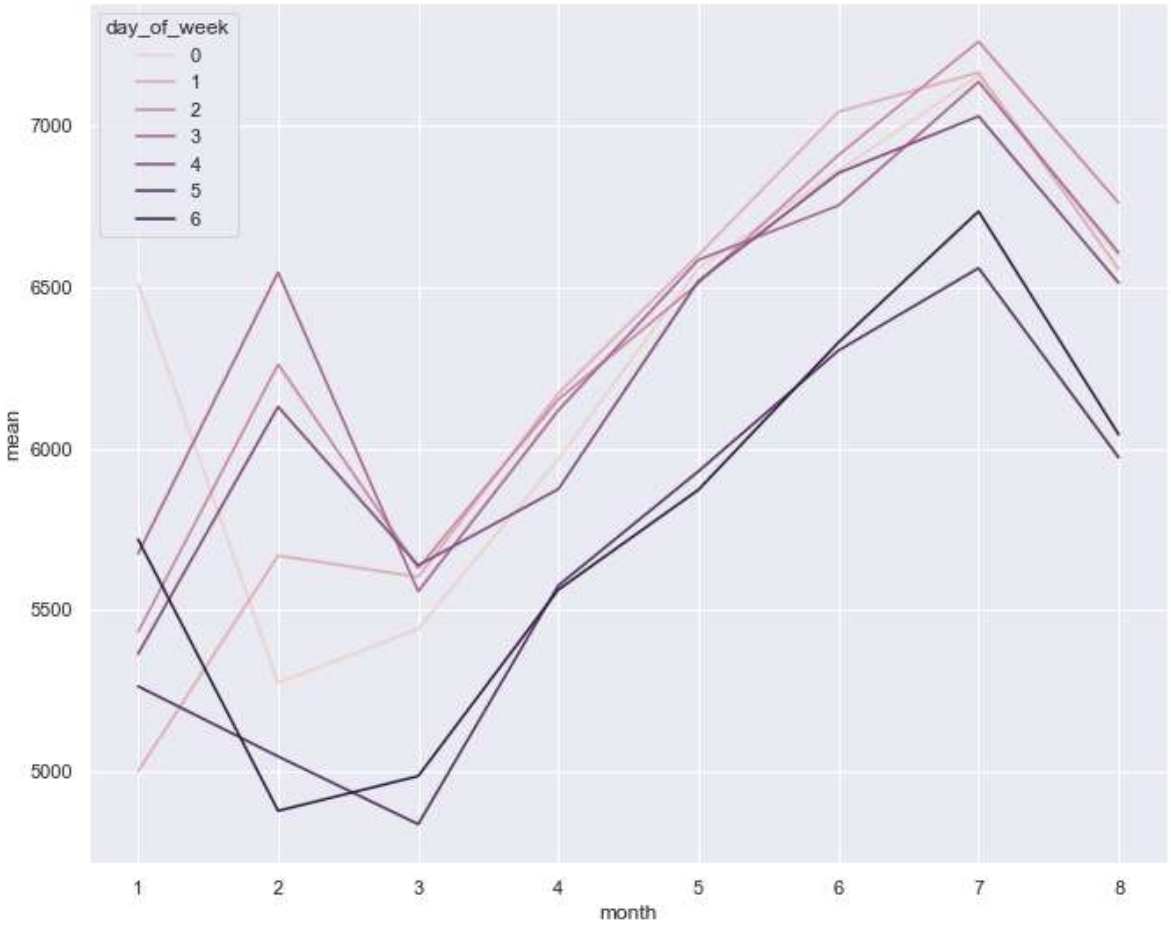
```
<seaborn.axisgrid.FacetGrid at 0x28800d33040>
```



The above facetgrid shows that the day of the week mays have some relationship with electricity consumption. It also shows a strong correlation between month and the price demand

In [24]:

```
sns.set(rc={"figure.figsize": (11,9)});
sns.lineplot(data=monthly_df,x="month", y="mean", hue='day_of_week', ci=None);
```



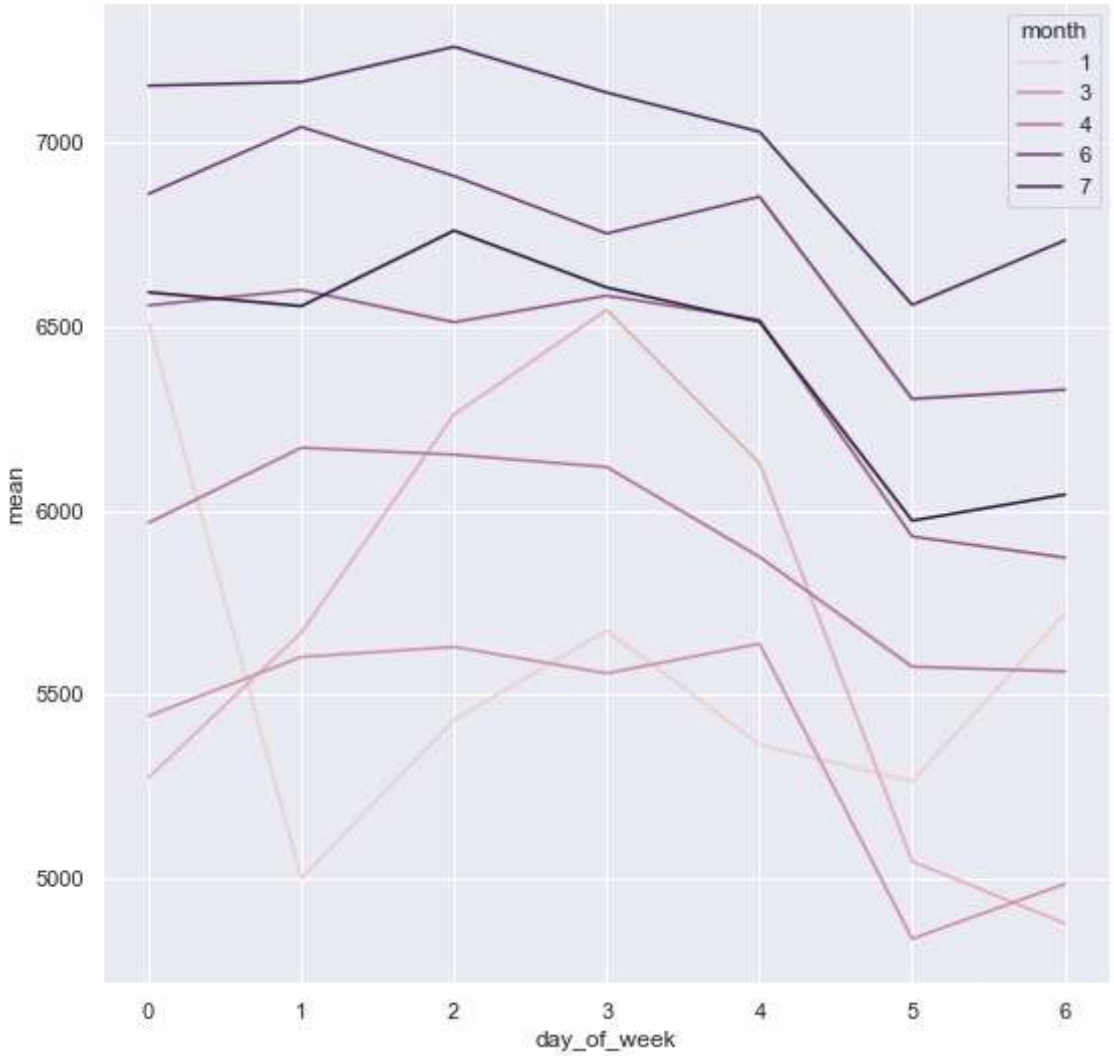
Lineplot demonstrates weekdays consume more Max energy than weekends for all months except January. This is due to abnormally huge demand on 11.01.2021 and 25.01.2021. While the spike on 25.01.21 is explainable as it is part of the long weekend ago and maybe associate with Australia day celebrations. On the other hand, the spike on 11.04.22 seems like an abberation. Hence, our stakeholders needs to be prepared for these anamolies.

Another anamoly in data is that weekend mean-max consumption dips in February whereas for weekdays it dips in March

The second change of trend for both weekend(0,6) and weekdays(0-4) is simultaneously in August where it breaks the increasing trend.

In [25]:

```
sns.set(rc={"figure.figsize": (9,9)});
sns.lineplot(data=monthly_df,x="day_of_week", y="mean", hue='month', ci=None);
```



As expected the day of the week seems to have a strong relationship with the electricity demand. The demand is clearly higher on weekdays then on weekends, on general. Specifically the demand usually peaks on Mondays and Tuesdays and move down from there on - for all months - except January and February.

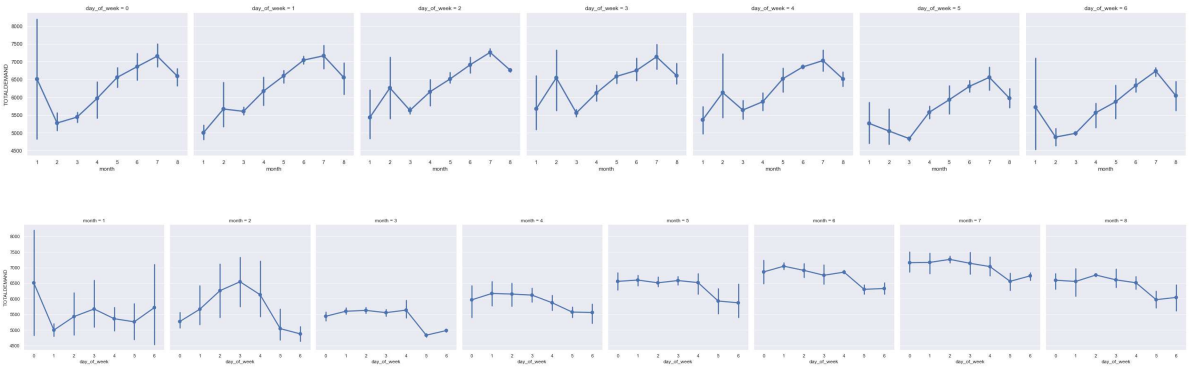
For our stakeholders - July seems to be a critical period at the max consumption for both weekdays and weekends peaks together, hence the max capacity and the base load for July will increase substantially as compared to other months

As we notice here that there are quite a few trends and anamolies in our data. Furthermore we have used ordinal encoding for three directions related features. To add to this, we only have 243 rows of data, which is exceptionally small dataframe and hence the standard train test split can yeild some very usual and unreliable results with high variance. Hence to manange this, we are going to use stratified K -fold for our regression analysis.

In [26]:

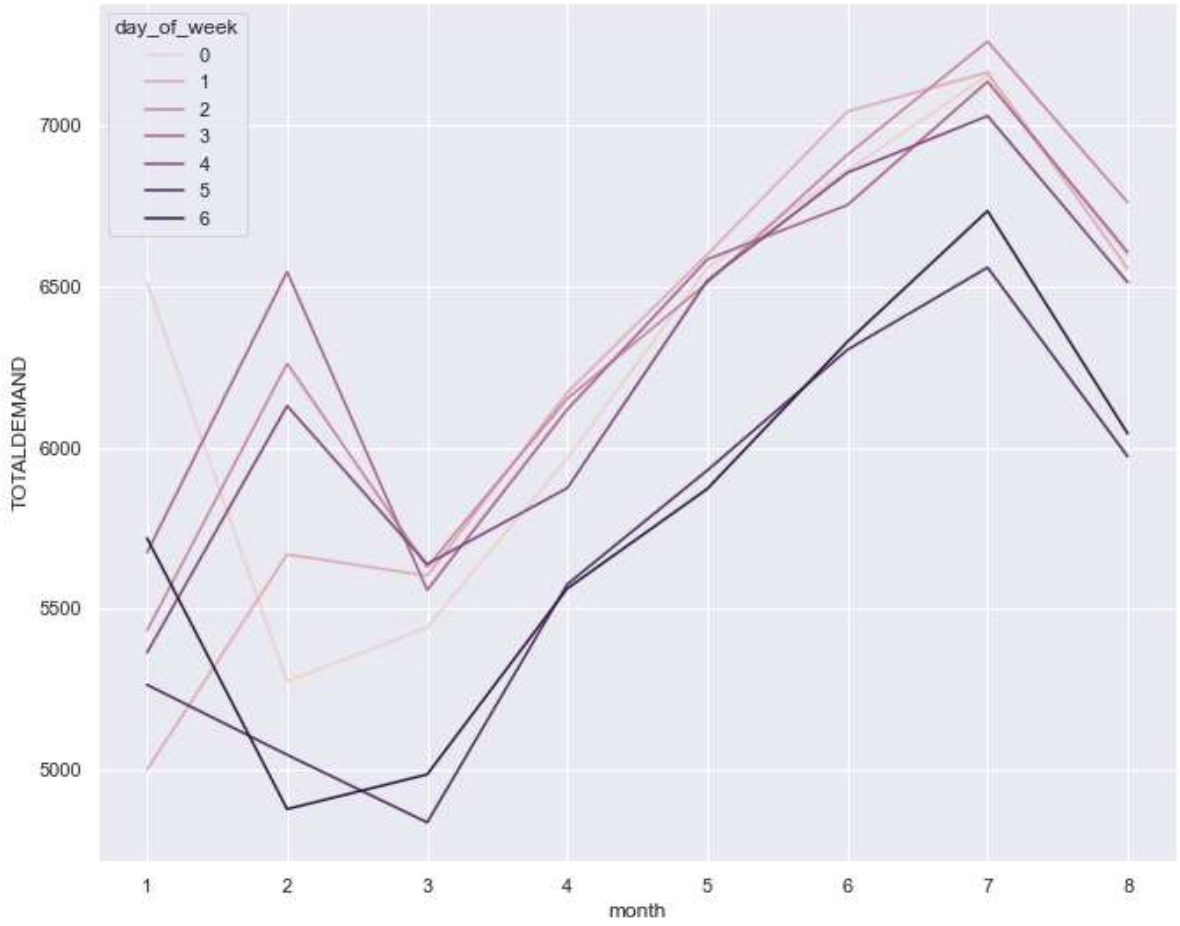
```
data = final_df.copy()
sns.set(rc = {'figure.figsize':(30,30)});
sns.factorplot(data=data,x="month", y="TOTALDEMAND", col='day_of_week');
sns.factorplot(data=data,x="day_of_week", y="TOTALDEMAND", col='month');
```

```
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\seaborn\categorica
l.py:3704: UserWarning: The `factorplot` function has been renamed to `catpl
ot`. The original name will be removed in a future release. Please update yo
ur code. Note that the default `kind` in `factorplot` (`'point'`) has change
d `'strip'` in `catplot`.
  warnings.warn(msg)
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\seaborn\categorica
l.py:3704: UserWarning: The `factorplot` function has been renamed to `catpl
ot`. The original name will be removed in a future release. Please update yo
ur code. Note that the default `kind` in `factorplot` (`'point'`) has change
d `'strip'` in `catplot`.
  warnings.warn(msg)
```



In [27]:

```
sns.set(rc = {'figure.figsize':(11,9)});
sns.lineplot(data=data,x="month", y="TOTALDEMAND", hue='day_of_week', ci=None);
```

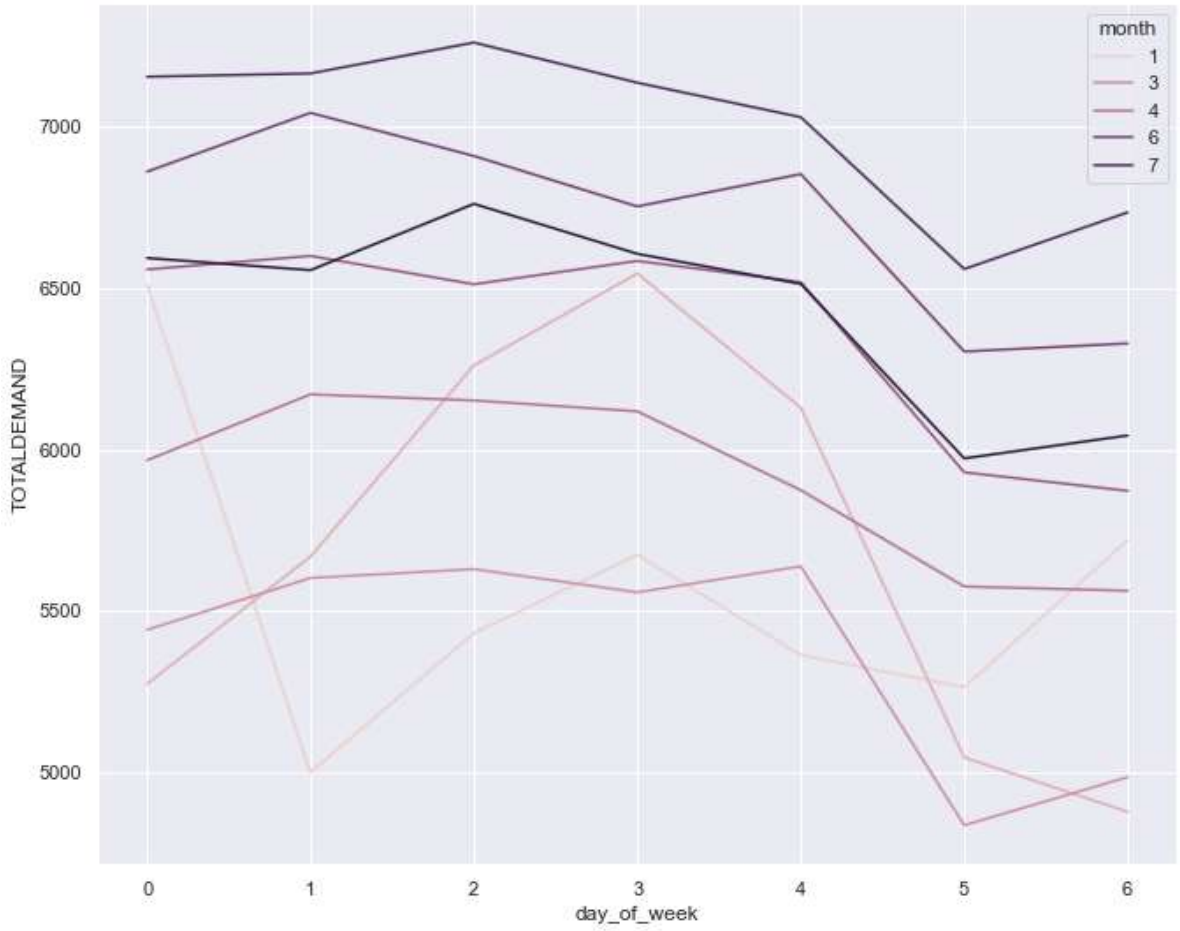


In [28]:

```
1 sns.lineplot(data=data,x="day_of_week", y="TOTALDEMAND", hue='month', ci=None)
```

Out[28]:

```
<AxesSubplot:xlabel='day_of_week', ylabel='TOTALDEMAND'>
```



Base Model - Linear Regression to predict Max Daily Energy

In [29]:

```
X = final_df[['month','Maximum temperature (°C)', 'Minimum temperature (°C)']]
y = y1 # y1 represents the column with total demand

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size = 0.
y_train = y_train.to_numpy()

# Generate the regression model (import .linear_model to use .LinearRegression() func)
linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_train_preds = linreg.predict(X_train)
y_test_preds = linreg.predict(X_test)

y_train_preds = y_train_preds.ravel()
y_test_preds = y_test_preds.ravel()

y_train = y_train.ravel()
y_test = y_test.to_numpy().ravel()
```

In [30]:

```
from sklearn.metrics import r2_score, mean_squared_error

r2_train = r2_score(y_train, y_train_preds)
r2_test = r2_score(y_test, y_test_preds)

mse_train = mean_squared_error(y_train, y_train_preds)
mse_test = mean_squared_error(y_test, y_test_preds)

print(f"R2 train: {r2_train} R2 test: {r2_test}")
print(f"MSE train: {mse_train: .3e} MSE test: {mse_test: .3e}")
```

```
R2 train: 0.39422004663916377 R2 test: 0.3099374249296186
MSE train:  3.922e+05 MSE test:  4.892e+05
```

In [31]:

```
# We are defining a function to calcuate adjusted r_2 values as it is not available unc

def adj_r2_score(r2, n_samples, n_regressors):
    factor = (n_samples - 1) / (n_samples - n_regressors - 1)
    return 1 - ((1 - r2) * factor)
```

In [32]:

```
adj_r2_train = adj_r2_score(r2_train, len(X_train), len(X_train.columns))
adj_r2_test = adj_r2_score(r2_test, len(X_test), len(X_test.columns))
print(f"Adj. R2 train: {adj_r2_train} Adj. R2 test: {adj_r2_test}")
```

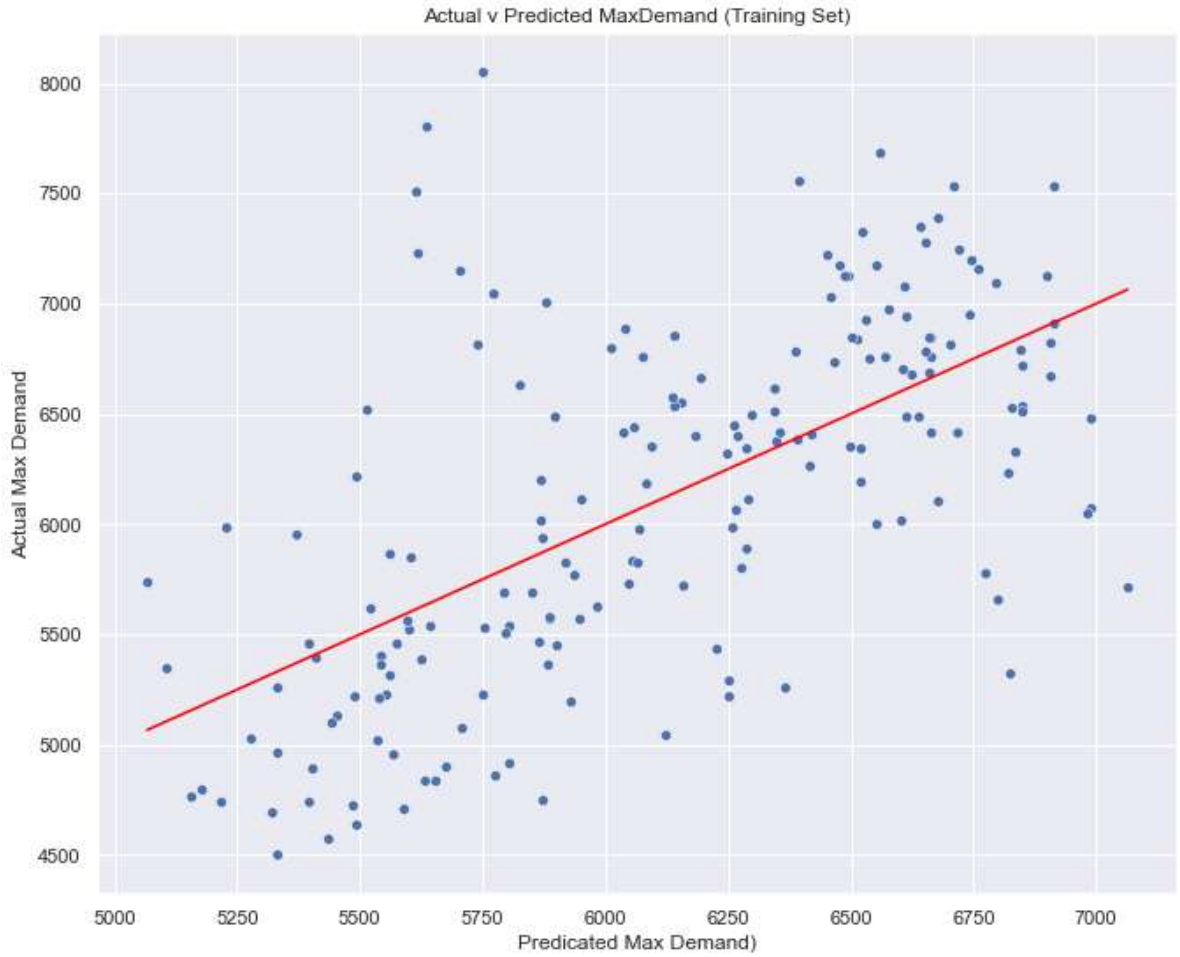
```
Adj. R2 train: 0.38401027214431827 Adj. R2 test: 0.2736183420311775
```

In [33]:

```
sns.scatterplot(x=y_train_preds, y=y_train)
sns.lineplot(x=y_train_preds, y=y_train_preds, color="red")
plt.title("Actual v Predicted MaxDemand (Training Set)")
plt.xlabel("Predicated Max Demand)")
plt.ylabel("Actual Max Demand")
```

Out[33]:

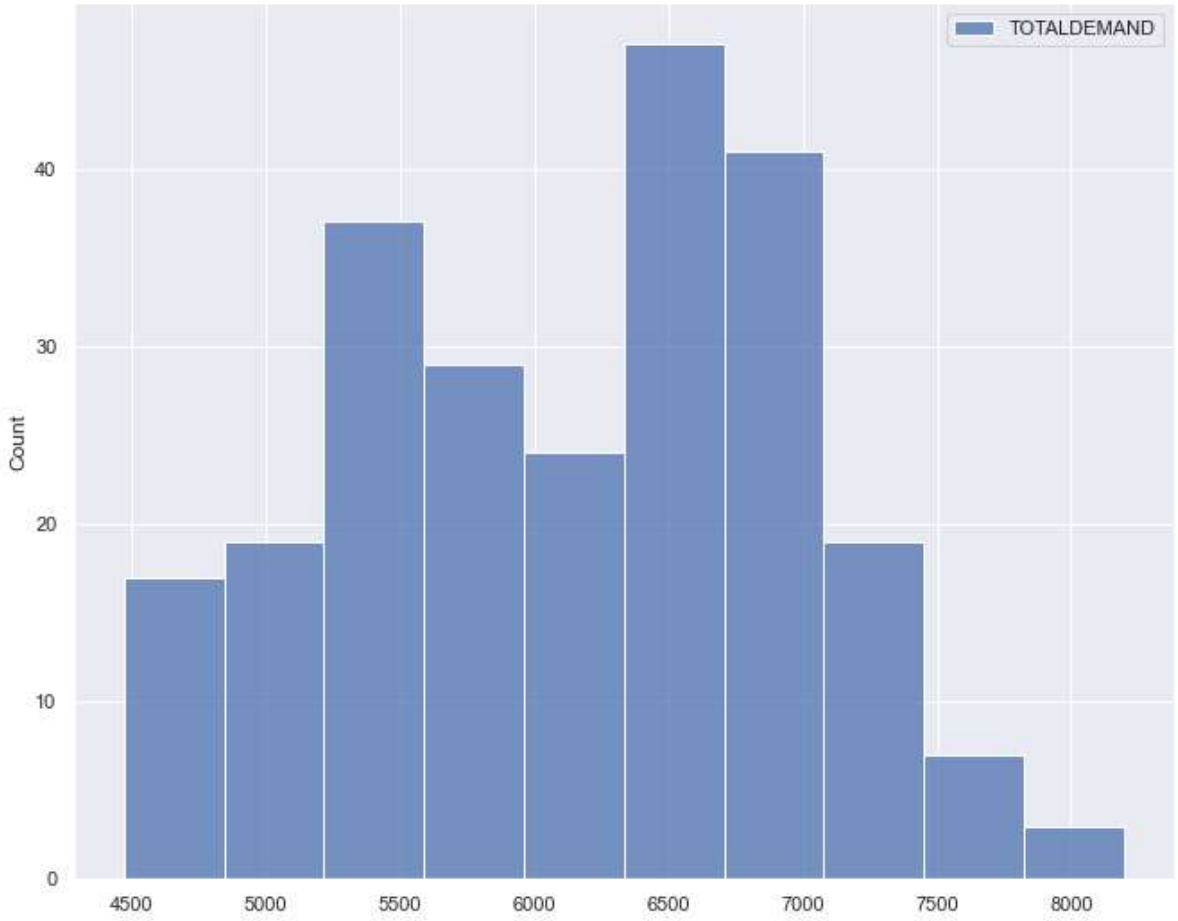
```
Text(0, 0.5, 'Actual Max Demand')
```



Here, we might want to normalize the dependant variable and then try fitting out our basemodel again

In [34]:

```
y = np.log1p(y1)
y = ((y - y.mean())/y.std()).astype(int)
sns.histplot(y1);
```



As compared to our original unscaled y values, our adj R2 test has increased from 0.27 to 0.32 and at the same time the MSE has decreased from 4.892e+05 to 0.0124. An extreme decrease.

Actually, this decrease in metrics is too high, although it is always possible. But it is good to do a stratified cross validation here, so as to ensure the quality of our results and confirms the extreme reduction in MSE score is not random or an aberration.

Stratified K Fold

In [35]:

```
def get_score(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)

    y_train_preds = linreg.predict(X_train)
    y_test_preds = linreg.predict(X_test)

    model_score = model.score(X_test, y_test)
    r2_train = r2_score(y_train, y_train_preds)
    r2_test = r2_score(y_test, y_test_preds)
    mean_sqrd_error_train = mean_squared_error(y_train, y_train_preds)
    mean_sqrd_error_test = mean_squared_error(y_test, y_test_preds)

    return [model_score, r2_train, r2_test, mean_sqrd_error_train, mean_sqrd_error_test
```

In [36]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score

def Average(lst):
    return sum(lst) / len(lst)

# this will help us to calculate 1 mean for every 5 values for each metric

scores = []
skf = StratifiedKFold(n_splits=5, random_state=None)
skf.get_n_splits(X,y)

# X is the feature set and y is the target

X = final_df[['month','Maximum temperature (°C)', 'Minimum temperature (°C)']]
y=y

linreg = LinearRegression()

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    score = get_score(linreg, X_train, X_test, y_train, y_test)
    scores.append(score)
    # the returned object from get_score function is a list of 5 elements


print(f'5 model scores are {scores[0]} and the average model score is {round(Average(sc
print(f'5 r2 train scores are {scores[1]} and the average r2 train score is {round(Aver
print(f'5 r2 test scores are {scores[2]} and the average r2 test score is {round(Averag
print(f'5 mean_sqrd_error_train scores are {scores[3]} and average mse train score is {
print(f'5 mean_sqrd_error_tests are {scores[4]} and the average mse test score is {roun


```

```
5 model scores are [-0.23490855622062945, 0.2710272651643386, -0.23490855622
062945, 0.32191324404741983, 0.474213114883557] and the average model score
is 0.12
5 r2 train scores are [0.28253895026150533, 0.20669656576811046, 0.282538950
26150533, 0.35032158244590295, 0.2755098241811296] and the average r2 train
score is 0.28
5 r2 test scores are [0.36762481725489016, 0.1828672334043162, 0.36762481725
489016, 0.3432148680610206, 0.29656578749312523] and the average r2 test sco
re is 0.31
5 mean_sqrd_error_train scores are [0.17897574151839224, 0.2385616754361836
7, 0.17897574151839224, 0.3224773906318396, 0.3748773957997618] and average
mse train score is 0.26
5 mean_sqrd_error_tests are [-0.8109091820523655, 0.356835182620473, -0.8109
091820523655, 0.27238727729335716, 0.8268561022218265] and the average mse t
est score is -0.03


C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_sele
ction\_split.py:684: UserWarning: The least populated class in y has only 3
members, which is less than n_splits=5.
  warnings.warn(
```

In [37]:

```
eval_list = list()
avg_score = dict()
avg_score["dataset"] = "baseline with stratified KFold"

avg_score["avg_model_score"] = round(Average(scores[0]),2)
avg_score["avg_r2_train"] = round(Average(scores[1]),2)
avg_score["avg_r2_test"] = round(Average(scores[2]),2)
avg_score["avg_mse_train"] = round(Average(scores[3]),2)
avg_score["avg_mse_test"] = round(Average(scores[4]),2)

avg_score["n_features"] = len(X.columns)

eval_list.append(avg_score)
pd.DataFrame(eval_list)
```

Out[37]:

	dataset	avg_model_score	avg_r2_train	avg_r2_test	avg_mse_train	avg_mse_test	n_featur
0	baseline with stratified KFold	0.12	0.28	0.31	0.26	-0.03	

Iteration : with all feats_pre transformation and scaling

In [38]:

```
X = final_df.drop(['Date','TOTALDEMAND', 'PRICECATEGORY'], axis=1)
y = y

scores = []
skf = StratifiedKFold(n_splits=5, random_state=None)
skf.get_n_splits(X,y)

linreg = LinearRegression()

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    score = get_score(linreg,X_train, X_test, y_train, y_test)
    scores.append(score)
    # the returned object from get_score function is a list of 5 elements


print(f'5 model scores are {scores[0]} and the average model score is {round(Average(sc
print(f'5 r2 train scores are {scores[1]} and the average r2 train score is {round(Aver
print(f'5 r2 test scores are {scores[2]} and the average r2 test score is {round(Averag
print(f'5 mean_sqrd_error_train scores are {scores[3]} and average mse train score is {
print(f'5 mean_sqrd_error_tests are {scores[4]} and the average mse test score is {roun

```

```
5 model scores are [-0.7081006327803174, 0.4813681083048176, -0.708100632780
3174, 0.22902704963263718, 0.6559220255824458] and the average model score i
s -0.01
5 r2 train scores are [0.18409849557058855, 0.4080527404268208, 0.1840984955
7058855, 0.26140300388208726, 0.31331161477880776] and the average r2 train
score is 0.27
5 r2 test scores are [0.36200661269501233, 0.3858510282298746, 0.36200661269
501233, 0.25795692809390325, 0.2992005639756002] and the average r2 test sco
re is 0.33
5 mean_sqrd_error_train scores are [0.2277072722874549, 0.4223844732704902,
0.2277072722874549, 0.24462644161609542, 0.3526267142159711] and average mse
train score is 0.3
5 mean_sqrd_error_tests are [-0.9438848325001892, 0.47157178011881606, -0.94
38848325001892, 0.22379508357571565, 0.887572414839496] and the average mse
test score is -0.06

C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_sele
ction\_split.py:684: UserWarning: The least populated class in y has only 3
members, which is less than n_splits=5.
  warnings.warn(
```

In [39]:

```
avg_score = dict()
avg_score["dataset"] = "all feats_pre transformation and scaling"

avg_score["avg_model_score"] = round(Average(scores[0]),2)
avg_score["avg_r2_train"] = round(Average(scores[1]),2)
avg_score["avg_r2_test"] = round(Average(scores[2]),2)
avg_score["avg_mse_train"] = round(Average(scores[3]),2)
avg_score["avg_mse_test"] = round(Average(scores[4]),2)

avg_score["n_features"] = len(X.columns)

eval_list.append(avg_score)
pd.DataFrame(eval_list)
```

Out[39]:

	dataset	avg_model_score	avg_r2_train	avg_r2_test	avg_mse_train	avg_mse_test	n_f
0	baseline with stratified KFold	0.12	0.28	0.31	0.26	-0.03	
1	all feats_pre transformation and scaling	-0.01	0.27	0.33	0.30	-0.06	

If you notice the results above, we have partly acheived our objective to make the test samples more representative of all the hidden trends and class types in our data.

This was acheived by increasing the number of features from 3 to 21.

In [40]:

```
continuous_feats = [ 'Minimum temperature (°C)', 'Maximum temperature (°C)', 'Rainfall
                     'Sunshine (hours)', 'Speed of maximum wind gust (km/h)',
       '9am Temperature (°C)', '9am relative humidity (%)',
       '9am cloud amount (oktas)', '9am wind speed (km/h)', '9am MSL pressure (hPa)',
       '3pm Temperature (°C)', '3pm relative humidity (%)',
       '3pm cloud amount (oktas)', '3pm wind speed (km/h)', '3pm MSL pressure (hPa)']

ordinal_dir = ['Direction of maximum wind gust ', '9am wind direction','3pm wind direct
time_feats = ['Date','month', 'day_of_week']
```

In [41]:

```
continuous_df = final_df[continuous_feats]
# continuous_df.hist(figsize=[20,20], bins='auto');
```

In [42]:

```
continuous_df.skew(axis=0)
```

Out[42]:

```
Minimum temperature (°C)              0.091033
Maximum temperature (°C)              1.107193
Rainfall (mm)                         5.447560
Evaporation (mm)                      1.062092
Sunshine (hours)                      0.245256
Speed of maximum wind gust (km/h)     0.560534
9am Temperature (°C)                  0.610747
9am relative humidity (%)            -0.279854
9am cloud amount (oktas)             -0.802511
9am wind speed (km/h)                 0.922306
9am MSL pressure (hPa)               -0.548846
3pm Temperature (°C)                  0.966494
3pm relative humidity (%)             0.229416
3pm cloud amount (oktas)             -0.893073
3pm wind speed (km/h)                 0.668156
3pm MSL pressure (hPa)               -0.474595
dtype: float64
```

In [43]:

```
skew_df = pd.DataFrame()
skew_df['original_skewness'] = continuous_df.skew(axis=0)
skew_df['log_transformed_skewness'] = np.log1p(continuous_df).skew(axis=0)
skew_df
```

Out[43]:

	original_skewness	log_transformed_skewness
Minimum temperature (°C)	0.091033	-0.892082
Maximum temperature (°C)	1.107193	0.506999
Rainfall (mm)	5.447560	1.949305
Evaporation (mm)	1.062092	-0.247032
Sunshine (hours)	0.245256	-0.681981
Speed of maximum wind gust (km/h)	0.560534	-0.268753
9am Temperature (°C)	0.610747	-0.529275
9am relative humidity (%)	-0.279854	-1.073606
9am cloud amount (oktas)	-0.802511	-1.215197
9am wind speed (km/h)	0.922306	-0.567753
9am MSL pressure (hPa)	-0.548846	-0.573964
3pm Temperature (°C)	0.966494	0.358444
3pm relative humidity (%)	0.229416	-0.567981
3pm cloud amount (oktas)	-0.893073	-1.440540
3pm wind speed (km/h)	0.668156	-0.590614
3pm MSL pressure (hPa)	-0.474595	-0.497042

As we can see in the skewness df that when we apply log transformed to all the features, then only for the features with skewness > 0.55 the skewness decreases. But for features with skewness < 0.55 the skewness increases significantly.

In orther words, we are going to apply log transformation to only features which are right skewed i.e. have a long right tail

In [44]:

```
# Applying log transformation to only right tailed data with skewness greater than 0.55

right_tailed = skew_df[continuous_df.skew(axis=0) > 0.55].index.to_list()
other_feats = skew_df[continuous_df.skew(axis=0) <= 0.55].index.to_list()

cont_log = np.log1p(continuous_df[right_tailed])
cont_log = pd.concat([cont_log, continuous_df[other_feats]], axis=1)
```

In [45]:

```
# Feature scaling to bring all features to the same scale and hence making all coeffici

norm_cols = list()
for c in cont_log.columns:
    norm_col = f"{c}_norm"
    cont_log[norm_col] = (cont_log[c] - cont_log[c].mean()) / cont_log[c].std()
    norm_cols.append(norm_col)
    cont_log.drop(axis=1, columns=c, inplace=True)
cont_log.head(3)
```

Out[45]:

	Maximum temperature (°C)_norm	Rainfall (mm)_norm	Evaporation (mm)_norm	Speed of maximum wind gust (km/h)_norm	9am Temperature (°C)_norm	9am wind speed (km/h)_norm	3pm Temperature (°C)_norm
0	1.819124	-0.585271	-0.181227	-0.166135	1.186060	-2.808045	1.840871
1	1.697978	-0.585271	1.592344	-0.267075	1.794341	1.270575	1.923350
2	1.296405	2.754179	0.563671	0.026614	1.036033	-1.645242	1.145503

In [46]:

```
preprocessed = pd.concat([cont_log, weather_df[ordinal_dir], weather_df[time_feats]], a
preprocessed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243 entries, 0 to 242
Data columns (total 22 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Maximum temperature (°C)_norm           243 non-null    float64
 1   Rainfall (mm)_norm                      243 non-null    float64
 2   Evaporation (mm)_norm                   243 non-null    float64
 3   Speed of maximum wind gust (km/h)_norm  243 non-null    float64
 4   9am Temperature (°C)_norm               243 non-null    float64
 5   9am wind speed (km/h)_norm              243 non-null    float64
 6   3pm Temperature (°C)_norm               243 non-null    float64
 7   3pm wind speed (km/h)_norm              243 non-null    float64
 8   Minimum temperature (°C)_norm           243 non-null    float64
 9   Sunshine (hours)_norm                   243 non-null    float64
 10  9am relative humidity (%)_norm          243 non-null    float64
 11  9am cloud amount (oktas)_norm           243 non-null    float64
 12  9am MSL pressure (hPa)_norm             243 non-null    float64
 13  3pm relative humidity (%)_norm          243 non-null    float64
 14  3pm cloud amount (oktas)_norm           243 non-null    float64
 15  3pm MSL pressure (hPa)_norm             243 non-null    float64
 16  Direction of maximum wind gust          243 non-null    float64
 17  9am wind direction                      243 non-null    float64
 18  3pm wind direction                      243 non-null    float64
 19  Date                                    243 non-null    datetime64[ns]
 20  month                                   243 non-null    int64
 21  day_of_week                             243 non-null    int64
dtypes: datetime64[ns](1), float64(19), int64(2)
memory usage: 41.9 KB
```

In [47]:

```
X = preprocessed.drop(columns='Date', axis=1)

scores = []
skf = StratifiedKFold(n_splits=5, random_state=None)
skf.get_n_splits(X,y)
# X is the feature set and y is the target
linreg = LinearRegression()

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    score = get_score(linreg,X_train, X_test, y_train, y_test)
    scores.append(score)
    # the returned object from get_score function is a list of 5 elements


print(f'5 model scores are {scores[0]} and the average model score is {round(Average(sc
print(f'5 r2 train scores are {scores[1]} and the average r2 train score is {round(Aver
print(f'5 r2 test scores are {scores[2]} and the average r2 test score is {round(Averag
print(f'5 mean_sqrd_error_train scores are {scores[3]} and average mse train score is {
print(f'5 mean_sqrd_error_tests are {scores[4]} and the average mse test score is {roun
```

```
5 model scores are [-0.7025026617897037, 0.47730911008359855, -0.7025026617897037, 0.23081949703503538, 0.653772367417787] and the average model score is -0.01
5 r2 train scores are [0.1646928133814083, 0.39542360012432676, 0.1646928133814083, 0.26698001291140633, 0.3207635260567854] and the average r2 train score is 0.26
5 r2 test scores are [0.3704968243285617, 0.36385236076500116, 0.3704968243285617, 0.2671968827990983, 0.29521889871138673] and the average r2 test score is 0.33
5 mean_sqrd_error_train scores are [0.22344136716341934, 0.4034869362476703, 0.22344136716341934, 0.2526297535481268, 0.35457451464586925] and average mse train score is 0.29
5 mean_sqrd_error_tests are [-0.8802314248364087, 0.4558459800776484, -0.8802314248364087, 0.2304551304886141, 0.8585084457152351] and the average mse test score is -0.04


C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_selection\_split.py:684: UserWarning: The least populated class in y has only 3 members, which is less than n_splits=5.
  warnings.warn(
```

In [48]:

```
avg_score = dict()
avg_score["dataset"] = "all feats post transformation and scaling"

avg_score["avg_model_score"] = round(Average(scores[0]),2)
avg_score["avg_r2_train"] = round(Average(scores[1]),2)
avg_score["avg_r2_test"] = round(Average(scores[2]),2)
avg_score["avg_mse_train"] = round(Average(scores[3]),2)
avg_score["avg_mse_test"] = round(Average(scores[4]),2)

avg_score["n_features"] = len(X.columns)

eval_list.append(avg_score)
pd.DataFrame(eval_list)
```

Out[48]:

	dataset	avg_model_score	avg_r2_train	avg_r2_test	avg_mse_train	avg_mse_test	n_f
0	baseline with stratified KFold	0.12	0.28	0.31	0.26	-0.03	
1	all feats_pre transformation and scaling	-0.01	0.27	0.33	0.30	-0.06	
2	all feats post transformation and scaling	-0.01	0.26	0.33	0.29	-0.04	

As we can see that our r_trainaverage is almost the same in all the three iterations of our model but there is a consistent trend of 0.33 average test r2 score with significant reduction in variability of the five test r2 scores. Also the mse for both train and trend data is consistently trending downwards from first to third iterations, which is a very good sign for our model.

Iteration: Feature Selection using f_regression

f_regression is a univariate linear regression tests returning F-statistics and p-values for each of the features one by one. f_regression is derived from r_regression and will rank features in the same order if all the features are positively correlated with the target.

Note however that contrary to f_regression, r_regression values lie in [-1, 1] and can thus be negative. f_regression is therefore recommended as a feature selection criterion to identify potentially predictive feature for a downstream classifier, irrespective of the sign of the association with the target variable.

In [49]:

```
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import SelectKBest

X = preprocessed.drop(columns='Date', axis=1)


# scores = []

skf = StratifiedKFold(n_splits=3, random_state=None)
skf.get_n_splits(X,y)


# X is the feature set and y is the target

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    names = pd.DataFrame(X_train.columns)

    model = SelectKBest(score_func = f_regression, k = 5)
    results = model.fit(X_train, y_train)

    results_df = pd.DataFrame(results.scores_)

    #Concat and name columns
    scored = pd.concat([names, results_df], axis = 1)
    scored.columns = ['Feature', 'Score']

    print(scored.sort_values(by = ['Score'], ascending = False))

    print("_____________________________________________________\n")
```

```
                                Feature      Score
19                                month  65.860597
6              3pm Temperature (°C)_norm  43.339398
8           Minimum temperature (°C)_norm  42.226714
4              9am Temperature (°C)_norm  34.401288
0           Maximum temperature (°C)_norm  31.466202
2                  Evaporation (mm)_norm  18.075075
20                          day_of_week  15.758247
17                 9am wind direction    9.482609
9                  Sunshine (hours)_norm   3.267703
18                 3pm wind direction    2.164770
10        9am relative humidity (%)_norm   2.078382
11         9am cloud amount (oktas)_norm   1.629483
5             9am wind speed (km/h)_norm   0.614549
13        3pm relative humidity (%)_norm   0.525088
1                  Rainfall (mm)_norm   0.283188
12           9am MSL pressure (hPa)_norm   0.237214
15           3pm MSL pressure (hPa)_norm   0.134330
7             3pm wind speed (km/h)_norm   0.089444
16       Direction of maximum wind gust    0.046849
3   Speed of maximum wind gust (km/h)_norm   0.031957
14          3pm cloud amount (oktas)_norm   0.006661
```

```
                                Feature      Score
```

	Feature	Score
19	month	16.708383
17	9am wind direction	16.154093
20	day_of_week	15.672344
8	Minimum temperature (°C)_norm	9.490296
4	9am Temperature (°C)_norm	7.269060
6	3pm Temperature (°C)_norm	5.621432
18	3pm wind direction	5.294876
0	Maximum temperature (°C)_norm	4.393610
5	9am wind speed (km/h)_norm	3.666169
15	3pm MSL pressure (hPa)_norm	3.115303
3	Speed of maximum wind gust (km/h)_norm	2.589301
1	Rainfall (mm)_norm	2.369793
10	9am relative humidity (%)_norm	2.254321
7	3pm wind speed (km/h)_norm	2.140915
11	9am cloud amount (oktas)_norm	1.976216
2	Evaporation (mm)_norm	1.213866
12	9am MSL pressure (hPa)_norm	0.870961
16	Direction of maximum wind gust	0.408515
14	3pm cloud amount (oktas)_norm	0.274856
9	Sunshine (hours)_norm	0.255653
13	3pm relative humidity (%)_norm	0.080672

	Feature	Score
19	month	52.007109
8	Minimum temperature (°C)_norm	16.833071
17	9am wind direction	16.444606
4	9am Temperature (°C)_norm	12.000853
11	9am cloud amount (oktas)_norm	10.716343
18	3pm wind direction	7.843528
16	Direction of maximum wind gust	6.343397
20	day_of_week	4.587409
2	Evaporation (mm)_norm	3.511936
12	9am MSL pressure (hPa)_norm	3.400576
3	Speed of maximum wind gust (km/h)_norm	2.049767
13	3pm relative humidity (%)_norm	1.883873
6	3pm Temperature (°C)_norm	1.777806
0	Maximum temperature (°C)_norm	1.395629
1	Rainfall (mm)_norm	1.136316
7	3pm wind speed (km/h)_norm	1.013272
10	9am relative humidity (%)_norm	0.447012
14	3pm cloud amount (oktas)_norm	0.369775
15	3pm MSL pressure (hPa)_norm	0.343334
9	Sunshine (hours)_norm	0.077317
5	9am wind speed (km/h)_norm	0.025068

```
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\va
lidation.py:1111: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), fo
r example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\va
lidation.py:1111: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), fo
r example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\va
lidation.py:1111: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), fo
```

```
r example using ravel().
  y = column_or_1d(y, warn=True)
```

As expected, we can see that the f_regression score for the features changes in the 3 n_splits. This is the reason to do split in a small dataset.

The features we choose, based on 3 iterations are: 'month', '3pm Temperature (°C)', 'Minimum temperature (°C)', '9am Temperature (°C)', 'Maximum temperature (°C)', '9am wind direction', 'day_of_week', 'Evaporation (mm)', '3pm wind direction', '9am cloud amount (oktas)'

In [50]:

```
X = preprocessed[['month', '3pm Temperature (°C)_norm', 'Minimum temperature (°C)_norm'
       'Maximum temperature (°C)_norm', '9am wind direction', 'day_of_week', 'Evaporati
       '3pm wind direction', '9am cloud amount (oktas)_norm' ]]

scores = []
skf = StratifiedKFold(n_splits=5, random_state=None)
skf.get_n_splits(X,y)
# X is the feature set and y is the target
linreg = LinearRegression()

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    score = get_score(linreg,X_train, X_test, y_train, y_test)
    scores.append(score)
    # the returned object from get_score function is a list of 5 elements


print(f'5 model scores are {scores[0]} and the average model score is {round(Average(sc
print(f'5 r2 train scores are {scores[1]} and the average r2 train score is {round(Aver
print(f'5 r2 test scores are {scores[2]} and the average r2 test score is {round(Averag
print(f'5 mean_sqrd_error_train scores are {scores[3]} and average mse train score is {
print(f'5 mean_sqrd_error_tests are {scores[4]} and the average mse test score is {roun
```

```
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_sele
ction\_split.py:684: UserWarning: The least populated class in y has only 3
members, which is less than n_splits=5.
  warnings.warn(


5 model scores are [-0.5003186943543607, 0.38930671657183935, -0.50031869435
43607, 0.269681219326603, 0.5761323765908873] and the average model score is
0.05
5 r2 train scores are [0.23856618134174912, 0.308621831389573, 0.23856618134
174912, 0.305311541138944, 0.29239566047601306] and the average r2 train sco
re is 0.28
5 r2 test scores are [0.40745093833950785, 0.27643076150683255, 0.4074509383
3950785, 0.30391599856786033, 0.27788848122853566] and the average r2 test s
core is 0.33
5 mean_sqrd_error_train scores are [0.20267219386109803, 0.3363622652744209,
0.20267219386109803, 0.28105777988220193, 0.3640576614835611] and average ms
e train score is 0.28
5 mean_sqrd_error_tests are [-0.6412282591348639, 0.3885761678329669, -0.641
2282591348639, 0.25894462572565163, 0.7493802641535923] and the average mse
test score is 0.02
```

In [51]:

```
avg_score = dict()
avg_score["dataset"] = "top 10 feats re Select K best, f_regression"

avg_score["avg_model_score"] = round(Average(scores[0]),2)
avg_score["avg_r2_train"] = round(Average(scores[1]),2)
avg_score["avg_r2_test"] = round(Average(scores[2]),2)
avg_score["avg_mse_train"] = round(Average(scores[3]),2)
avg_score["avg_mse_test"] = round(Average(scores[4]),2)

avg_score["n_features"] = len(X.columns)

eval_list.append(avg_score)
pd.DataFrame(eval_list)
```

Out[51]:

	dataset	avg_model_score	avg_r2_train	avg_r2_test	avg_mse_train	avg_mse_test	n_f
0	baseline with stratified KFold	0.12	0.28	0.31	0.26	-0.03	
1	all feats_pre transformation and scaling	-0.01	0.27	0.33	0.30	-0.06	
2	all feats post transformation and scaling	-0.01	0.26	0.33	0.29	-0.04	
3	top 10 feats re Select K best, f_regression	0.05	0.28	0.33	0.28	0.02	

Not surprisingly, our model has significantly improved. We have our highest r2_train and r2_test scores as well as minimum mse_test score till now, with only 10 features.

Iteration: Polynomials and interaction features

In [52]:

```
feature_names_in_ = [ 'month', 'Maximum temperature (°C)_norm', 'Minimum temperature (°
# Notice we have selected these features for interactions and polynomials as these 3 ha
# high f_regression scores in all three iteration and we have also tried them out in ba

X = preprocessed[feature_names_in_]

from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(include_bias=False, interaction_only=False)
poly_array = poly.fit_transform(X)
# poly.get_feature_names()
poly.get_feature_names_out(input_features= None)

poly_df = pd.DataFrame(poly_array, columns=poly.get_feature_names_out(input_features= N
```

In [53]:

```
poly_df.head()
```

Out[53]:

	month	Maximum temperature (°C)_norm	Minimum temperature (°C)_norm	month^2	month Maximum temperature (°C)_norm	month Minimum temperature (°C)_norm	Maximum temperature (°C)_norm^2	Maxim tempera (°C)_n Minim tempera (°C)_n
0	1.0	1.819124	1.177794	1.0	1.819124	1.177794	3.309214	2.142
1	1.0	1.697978	1.902762	1.0	1.697978	1.902762	2.883128	3.230
2	1.0	1.296405	1.540278	1.0	1.296405	1.540278	1.680666	1.996
3	1.0	-0.046622	1.281361	1.0	-0.046622	1.281361	0.002174	-0.059
4	1.0	0.056620	1.255469	1.0	0.056620	1.255469	0.003206	0.071

In [54]:

```
# Now we can proceed to check the significance of these interactive & polynomial featur

X=poly_df.drop(['month', 'Maximum temperature (°C)_norm', 'Minimum temperature (°C)_nor
X=pd.concat([poly_df,preprocessed.drop(columns='Date')], axis=1)


scores = []
skf=StratifiedKFold(n_splits=5, random_state=None)
skf.get_n_splits(X,y)

# X is the feature set and y is the target
linreg=LinearRegression()

for train_index, test_index in skf.split(X,y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    score = get_score(linreg,X_train, X_test, y_train, y_test)
    scores.append(score)

    # the returned object from get_score function is a list of 5 elements



print(f'5 model scores are {scores[0]} and the average model score is {round(Average(sc
print(f'5 r2 train scores are {scores[1]} and the average r2 train score is {round(Aver
print(f'5 r2 test scores are {scores[2]} and the average r2 test score is {round(Averag
print(f'5 mean_sqrd_error_train scores are {scores[3]} and average mse train score is {
print(f'5 mean_sqrd_error_tests are {scores[4]} and the average mse test score is {roun
```

```
C:\Users\udhai\anaconda3\envs\learn-env\lib\site-packages\sklearn\model_sele
ction\_split.py:684: UserWarning: The least populated class in y has only 3
members, which is less than n_splits=5.
  warnings.warn(


5 model scores are [-0.05437489760906877, 0.6837770442427562, -0.05437489760
906877, 0.13964357329911234, 0.4048869869202671] and the average model score
is 0.22
5 r2 train scores are [0.23581677631123144, 0.6416752109744444, 0.2358167763
1123144, 0.15823567843566622, 0.2934514503294647] and the average r2 train s
core is 0.31
5 r2 test scores are [0.7545574950716325, 0.5567760290504347, 0.754557495071
6325, 0.1861644312033885, 0.11510548127835973] and the average r2 test score
is 0.47
5 mean_sqrd_error_train scores are [0.2962297410065502, 0.6422938531109288,
0.2962297410065502, 0.15149243364895737, 0.32133954533902304] and average ms
e train score is 0.34
5 mean_sqrd_error_tests are [0.2646325151833916, 0.6500655111532602, 0.26463
25151833916, 0.14820105216010254, 0.3357667508798055] and the average mse te
st score is 0.33
```

In [55]:

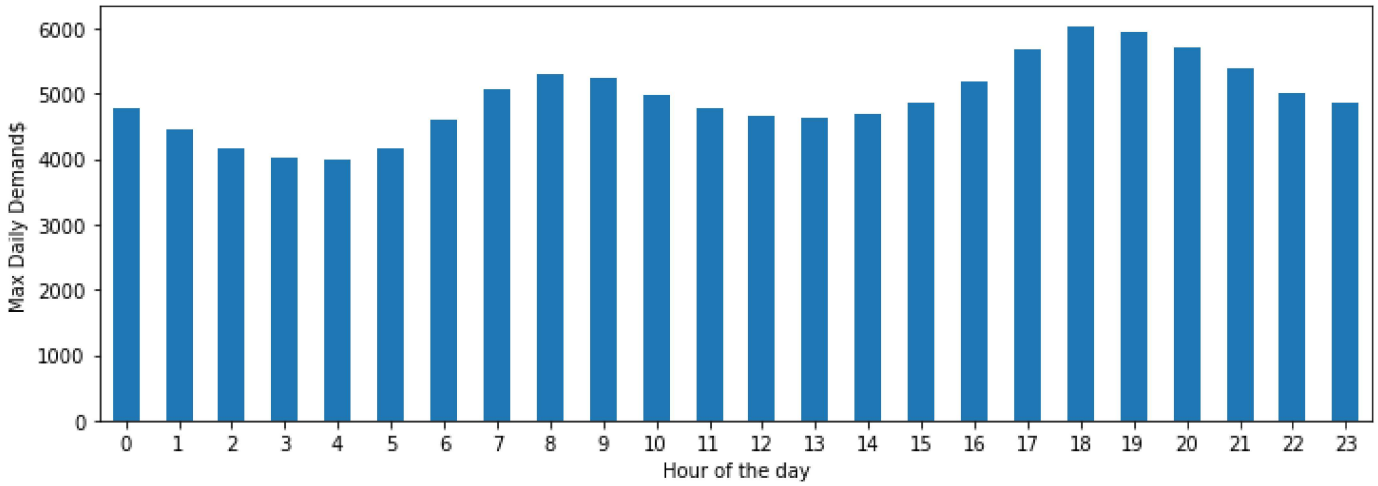
```
avg_score = dict()
avg_score["dataset"] = "all scaled feats plus polynomials & interactive feats"

avg_score["avg_model_score"] = round(Average(scores[0]),2)
avg_score["avg_r2_train"] = round(Average(scores[1]),2)
avg_score["avg_r2_test"] = round(Average(scores[2]),2)
avg_score["avg_mse_train"] = round(Average(scores[3]),2)
avg_score["avg_mse_test"] = round(Average(scores[4]),2)

avg_score["n_features"] = len(X.columns)

eval_list.append(avg_score)
pd.DataFrame(eval_list)
```

Out[55]:

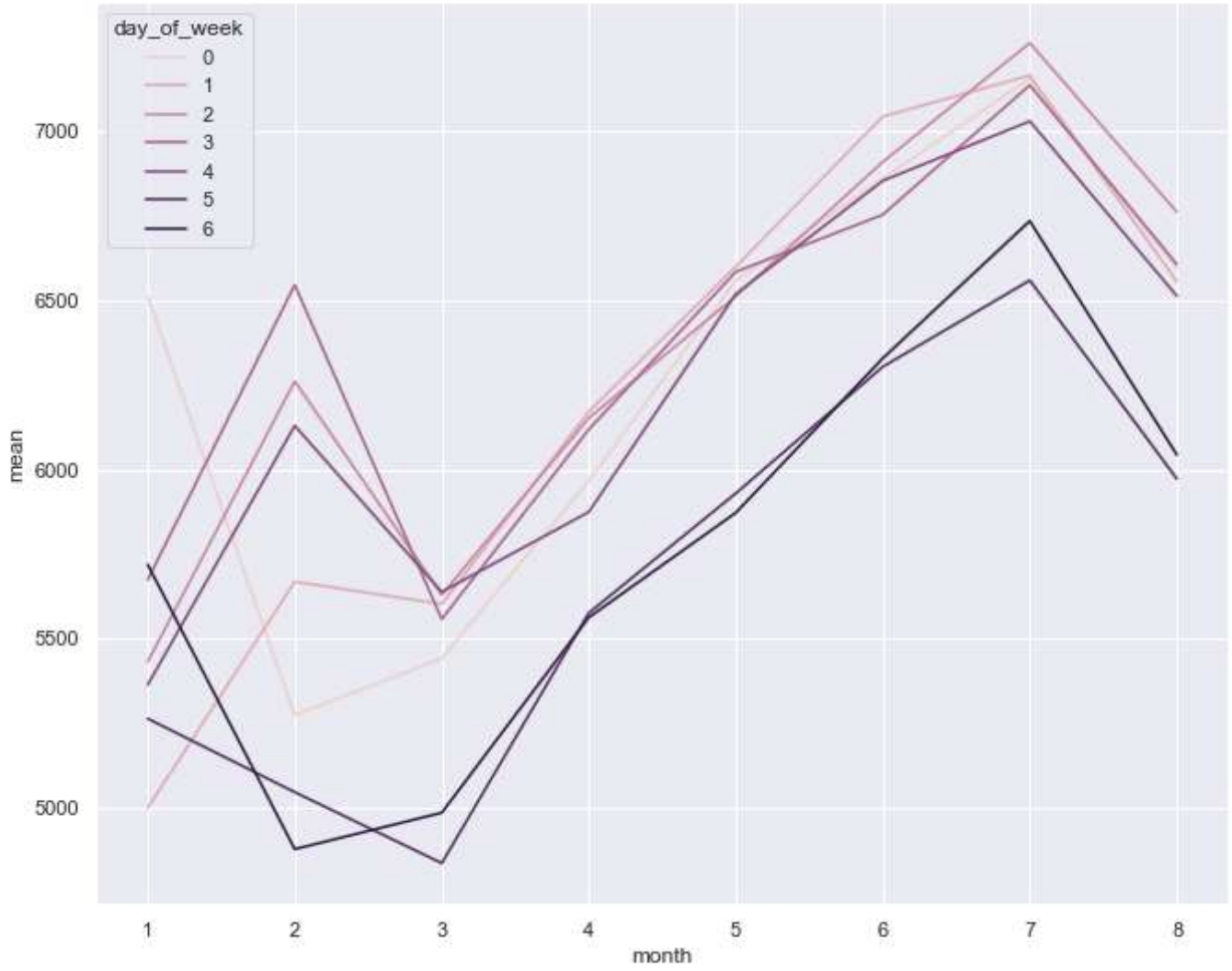
	dataset	avg_model_score	avg_r2_train	avg_r2_test	avg_mse_train	avg_mse_test	n_f
0	baseline with stratified KFold	0.12	0.28	0.31	0.26	-0.03	
1	all feats_pre transformation and scaling	-0.01	0.27	0.33	0.30	-0.06	
2	all feats post transformation and scaling	-0.01	0.26	0.33	0.29	-0.04	
3	top 10 feats re Select K best, f_regression	0.05	0.28	0.33	0.28	0.02	
4	all scaled feats plus polynomials & interactiv...	0.22	0.31	0.47	0.34	0.33	

As expected, the model with polynomial and interactive features have significantly improved our r2_test scores. Also the average model fit has significantly increased. So we will keep these interactive features and polynomials in our final model.

1. As expected, the hour of the day matters even on a eight month span. Electricity consumption is maximum at 6pm in the evening. Whereas the mean hourly electricity consumption dips to a minimum at 3 am. So this should inform at what time most generators come on and off the grid. This should inform the hourly forecast of the retail distributors.

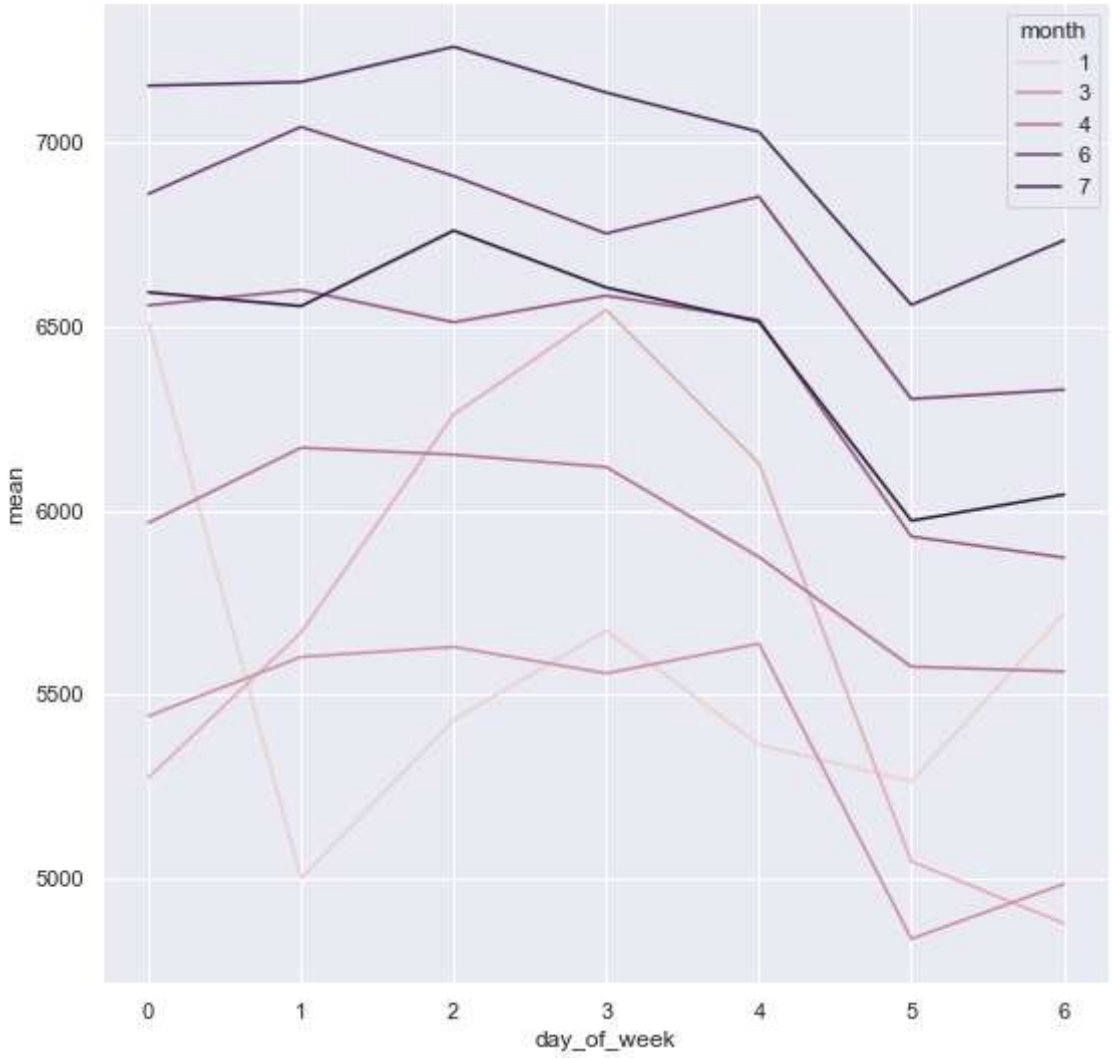


2. Lineplot demonstrates weekdays consume more Max energy than weekends for all months except January. This is due to abnormally huge demand on 11.01.2021 and 25.01.2021. While the spike on 25.01.21 is explainable as it is part of the long weekend ago and maybe associate with Australia day celebrations. On the other hand, the spike on 11.04.22 seems like an abberation. Hence, our stakeholders needs to be prepared for these anamolies.



Another anamoly in data is that weekend mean-max consumption dips in February whereas for weekdays it dips in March

3. For our stakeholders - July seems to be a critical period at the max consumption for both weekdays and weekends peaks together, hence the max capacity and the base load for July will increase substantially as compared to other months



4. Our final model with scaled, polynomials and interaction features is performing well and is able to explain 0.47 of variations in unseen/test data.

	dataset	avg_model_score	avg_r2_train	avg_r2_test	avg_mse_train	avg_mse_test	n_features
0	baseline	0.12	0.28	0.31	0.26	-0.03	3
1	all feats_pre transformation and scaling	-0.01	0.27	0.33	0.30	-0.06	21
2	all feats post transformation and scaling	-0.01	0.26	0.33	0.29	-0.04	21
3	top 10 feats re Select K best, f_regression	0.05	0.28	0.33	0.28	0.02	10
4	all scaled feats plus polynomials and interact...	0.22	0.31	0.47	0.34	0.33	30