Modèles de Machine Learning

Implémentation de différents modèles de machine learning appliqués à la prédiction de données sur les données de consommation électrique.

Import des librairies nécessaires, initialisation et mise en forme du set de données.

On importe les différentes librairies Python nécessaires : Pandas / Numpy / Seaborn / XGBoost.

In [1]:
```
#importation des librairies nécéssaires
import pandas as pd
import numpy as np
import seaborn as sns
import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error

dataset = pd.read_csv('DAYTON_hourly.csv', index_col='Datetime')
dataset1 = pd.read_csv('OHDAYTON.csv', index_col='Datetime')
dataset.index = pd.to_datetime(dataset.index)
dataset1.index = pd.to_datetime(dataset1.index)
```

In [2]:
```
#info du dataset
dataset.info()
dataset.shape
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 119019 entries, 2005-12-31 01:00:00 to 2018-01-02 00:00:00
Data columns (total 1 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   DAYTON_MW  119019 non-null  float64
dtypes: float64(1)
memory usage: 1.8 MB
```

Out[2]: (119019, 1)

In [3]:
```
#info du dataset
dataset1.info()
dataset1.shape
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4960 entries, 2005-01-01 to 2018-07-31
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Température  4960 non-null   float64
dtypes: float64(1)
memory usage: 77.5 KB
```

Out[3]: (4960, 1)

On met en forme de set de données sur lequel nous travaillons.

In [4]:
```
#comptage du nombre de données manquantes dans le dataset
dataset.isnull().sum()
```

Out[4]:
```
DAYTON_MW    0
dtype: int64
```

In [5]:
```
#comptage du nombre de données manquantes dans le dataset
dataset1.isnull().sum()
```

Out[5]:
```
Température    0
dtype: int64
```

In [6]:
```
#résumé statistique
dataset.describe()
```

Out[6]:

	DAYTON_MW
count	119019.000000
mean	2038.759080
std	394.785248
min	982.000000
25%	1749.000000
50%	2009.000000
75%	2281.000000
max	3746.000000

In [7]:
```
#résumé statistique
dataset1.describe()
```

Out[7]:

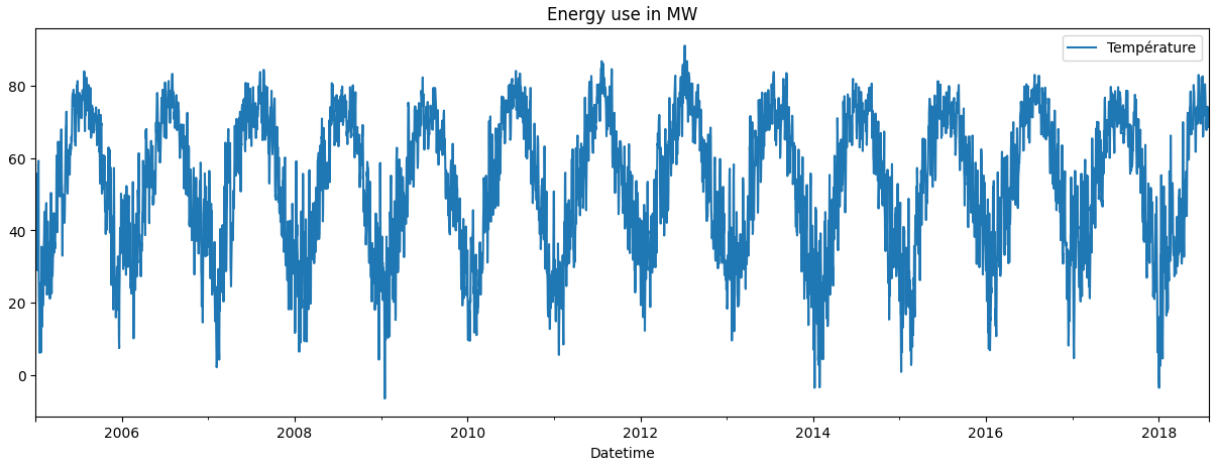
	Température
count	4960.000000
mean	52.886815
std	18.948147
min	-6.600000
25%	37.600000
50%	55.200000
75%	69.700000
max	91.200000

In [8]:
```
dataset = dataset.sort_values(by="Datetime")
```

In [9]:
```
dataset1 = dataset1.sort_values(by="Datetime")
```

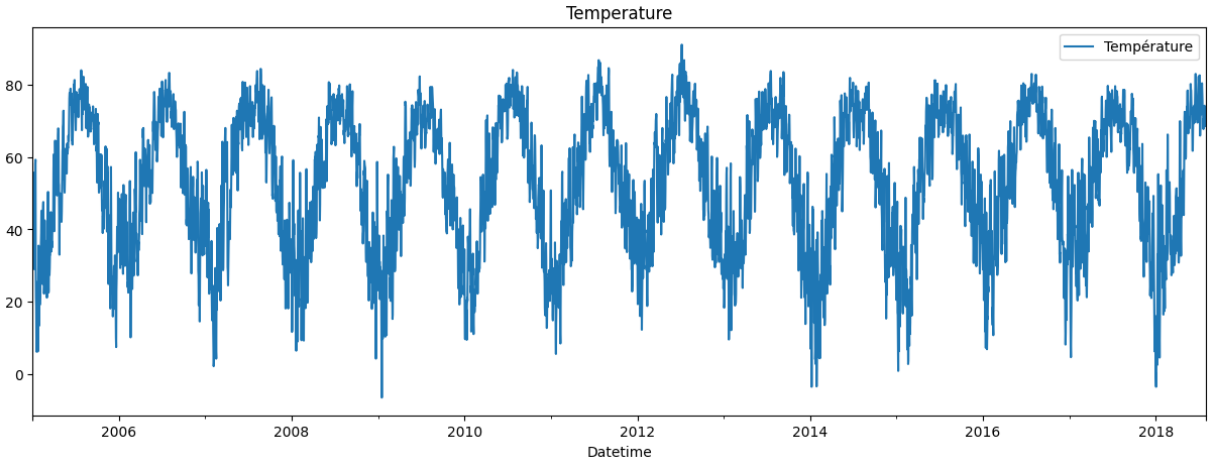
In [10]:
```
# plot des données
dataset1.plot(figsize=(15,5),title='Energy use in MW')
```

Out[10]:
```
<AxesSubplot: title={'center': 'Energy use in MW'}, xlabel='Datetime'>
```



In [11]:
```
dataset1.plot(figsize=(15,5),title='Temperature')
```

Out[11]:
```
<AxesSubplot: title={'center': 'Temperature'}, xlabel='Datetime'>
```



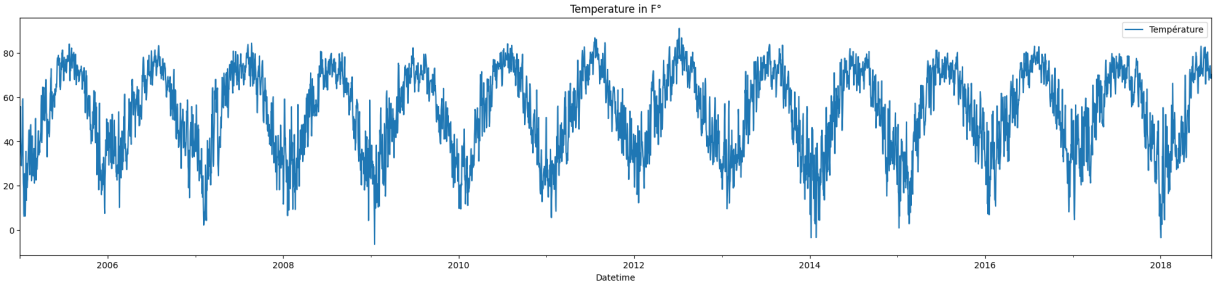
In [12]:
```
daily_groups = dataset.resample('D')
```

In [13]:
```
dataset_daily = daily_groups.sum()
```

Séparation du set de données.

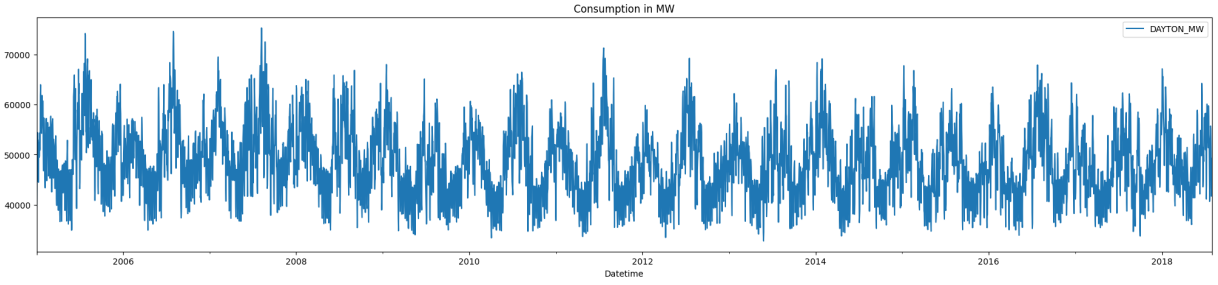
In [14]:
```
# plot des données
dataset1.plot(figsize=(25,5),title='Temperature in F°')
```

Out[14]: <AxesSubplot: title={'center': 'Temperature in F°'}, xlabel='Datetime'>



In [15]:
```
# plot des données
dataset_daily.plot(figsize=(25,5),title='Consumption in MW')
```

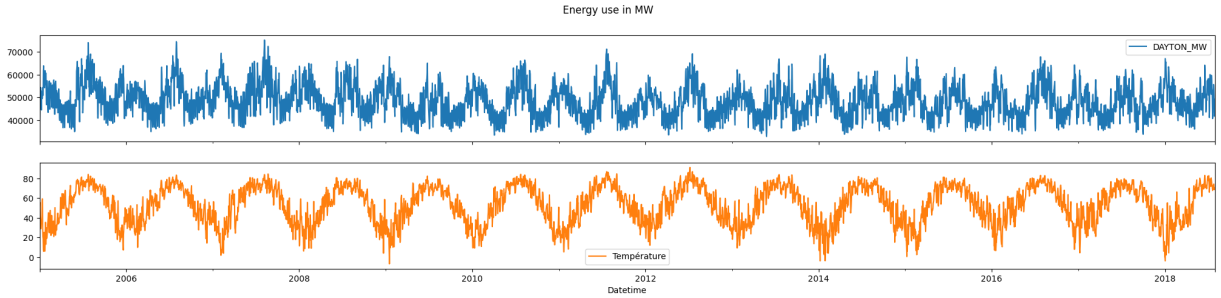
Out[15]: <AxesSubplot: title={'center': 'Consumption in MW'}, xlabel='Datetime'>



In [16]:
```
dataset_final=dataset_daily.join(dataset1,on='Datetime')
```

In [17]:
```
dataset_final.plot(figsize=(25,5),title='Energy use in MW',subplots='True')
```

Out[17]:
```
array([<AxesSubplot: xlabel='Datetime'>, <AxesSubplot: xlabel='Datetime'>],
      dtype=object)
```

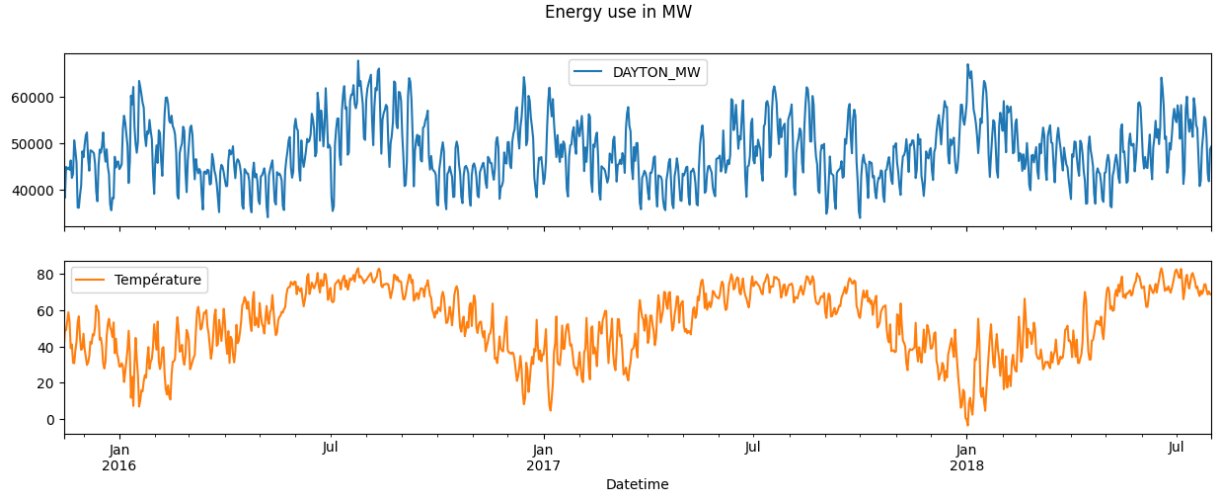


Le set de données est séparé en 2 parties, l'une appelée "**Train**" contenant 80 % des données du set et l'autre appelée "**Test**".

In [18]:
```
#split train et test
nb_lines = dataset_final.shape[0]
train = dataset_final.iloc[:int(nb_lines*0.8)]
test = dataset_final.iloc[int(nb_lines*0.8)+1:]
```

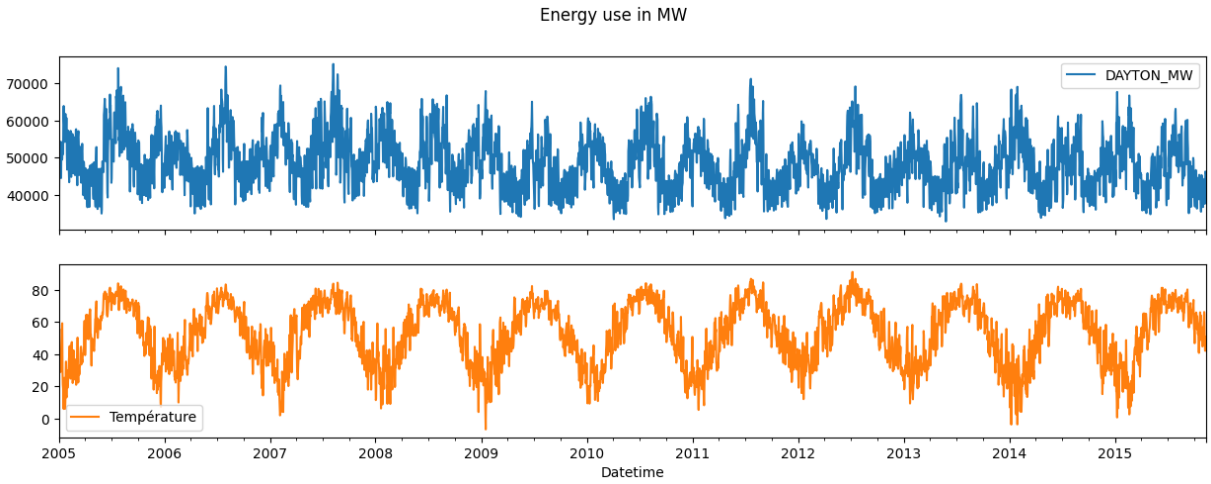
In [19]:
```
test.plot(figsize=(15,5),title='Energy use in MW',subplots='True')
```

Out[19]:
```
array([<AxesSubplot: xlabel='Datetime'>, <AxesSubplot: xlabel='Datetime'>],
      dtype=object)
```



In [20]:
```
train.plot(figsize=(15,5),title='Energy use in MW',subplots='True')
```

Out[20]:
```
array([<AxesSubplot: xlabel='Datetime'>, <AxesSubplot: xlabel='Datetime'>],
      dtype=object)
```



Création d'une fonction qui explicite l'heure, le jour, la semaine, le mois et l'année pour chaque données présente dans notre set de données.

In [21]:
```
def creation_index_temps(ds):
    ds['Jour(Semaine)']=ds.index.dayofweek
    ds['Semaine']=ds.index.week
    ds['Mois']=ds.index.month
    ds['Année']=ds.index.year
    ds['Jour(Année)']=ds.index.day
    return ds
```

In [22]:
```
creation_index_temps(dataset_final)
```

```
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:3: FutureWarn
ing: weekofyear and week have been deprecated, please use DatetimeIndex.isoca
lendar().week instead, which returns a Series. To exactly reproduce the behav
ior of week and weekofyear and return an Index, you may call pd.Int64Index(id
x.isocalendar().week)
  ds['Semaine']=ds.index.week
```

Out[22]:

Datetime	DAYTON_MW	Température	Jour(Semaine)	Semaine	Mois	Année	Jour(Année
2005-01-01	37036.0	48.1	5	53	1	2005	
2005-01-02	39528.0	50.9	6	53	1	2005	
2005-01-03	47581.0	55.8	0	1	1	2005	
2005-01-04	50831.0	42.5	1	1	1	2005	
2005-01-05	54399.0	36.3	2	1	1	2005	
...	...	...	...	...	...	...	.
2018-07-27	51037.0	71.0	4	30	7	2018	2
2018-07-28	43442.0	68.7	5	30	7	2018	2
2018-07-29	41731.0	70.3	6	30	7	2018	2
2018-07-30	48632.0	68.8	0	31	7	2018	3
2018-07-31	49226.0	68.7	1	31	7	2018	3

4960 rows × 7 columns

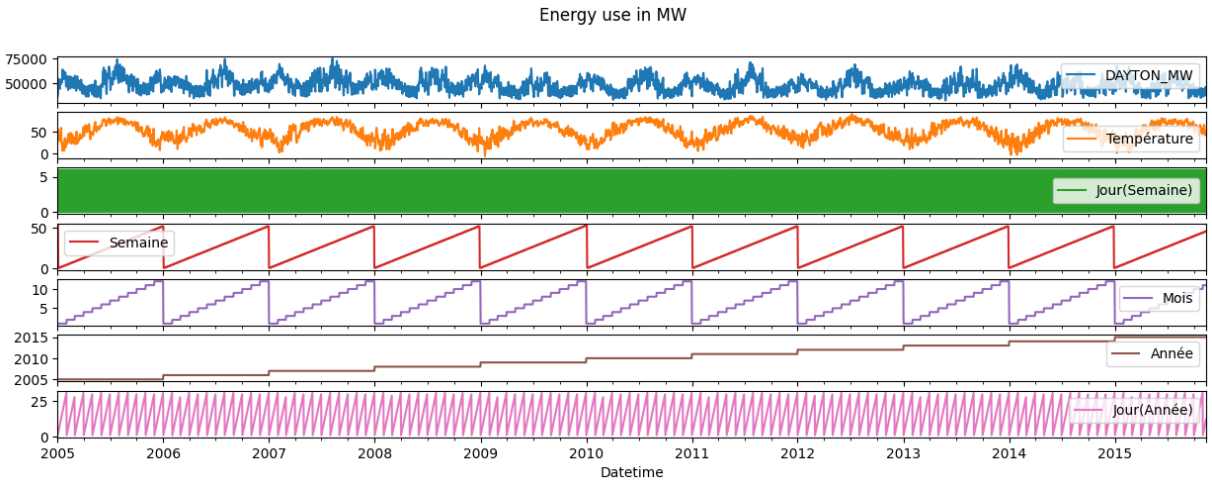
In [23]:
```
train = creation_index_temps(train)
test = creation_index_temps(test)
```

```
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Jour(Semaine)']=ds.index.dayofweek
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:3: FutureWarn
ing: weekofyear and week have been deprecated, please use DatetimeIndex.isoca
lendar().week instead, which returns a Series. To exactly reproduce the behav
ior of week and weekofyear and return an Index, you may call pd.Int64Index(id
x.isocalendar().week)
  ds['Semaine']=ds.index.week
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:3: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Semaine']=ds.index.week
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:4: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Mois']=ds.index.month
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:5: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Année']=ds.index.year
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:6: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Jour(Année)']=ds.index.day
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Jour(Semaine)']=ds.index.dayofweek
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:3: FutureWarn
ing: weekofyear and week have been deprecated, please use DatetimeIndex.isoca
```

```
lendar().week instead, which returns a Series. To exactly reproduce the behav
ior of week and weekofyear and return an Index, you may call pd.Int64Index(id
x.isocalendar().week)
  ds['Semaine']=ds.index.week
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:3: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Semaine']=ds.index.week
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:4: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Mois']=ds.index.month
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:5: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Année']=ds.index.year
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2549928062.py:6: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  ds['Jour(Année)']=ds.index.day
```

In [24]:
```
train.plot(figsize=(15,5),title='Energy use in MW',subplots='True')
```

Out[24]:
```
array([<AxesSubplot: xlabel='Datetime'>, <AxesSubplot: xlabel='Datetime'>,
       <AxesSubplot: xlabel='Datetime'>, <AxesSubplot: xlabel='Datetime'>,
       <AxesSubplot: xlabel='Datetime'>, <AxesSubplot: xlabel='Datetime'>,
       <AxesSubplot: xlabel='Datetime'>], dtype=object)
```



Entraînement sur le set de données "**Train**".

On liste toutes les entrées sur lesquelles nous allons appliquer l'algorithme d'entraînement, ainsi que la sortie souhaitée (la valeur que nous souhaitons prédire avec le modèle).

- SANS TEMPERATURE

In [25]:
```
ENTREES = ['Jour(Semaine)', 'Semaine', 'Mois', 'Année', 'Jour(Année)']
SORTIES = ['DAYTON_MW']
```

In [26]:
```
X_train = train[ENTREES]
Y_train = train[SORTIES]

X_test = test[ENTREES]
Y_test = test[SORTIES]
```

On entraîne le modèle de machine learning sur le set de données "**Train**" en utilisant la méthode de boosting de gradient extrême (XGBoost).

In [27]:
```
reg = xgb.XGBRegressor(
    n_estimators=200000,         # Hauteur maximale de l'arbre de décision cr
    early_stopping_rounds=200,   # Nombre maximal de sorties de la fonction
    learning_rate=0.0001)        # Ratio d'apprentissage
reg.fit(
    X_train, Y_train,
    eval_set=[(X_train, Y_train),(X_test,Y_test)],
    verbose=1000)               # Pas d'affichage des sorties (ici toutes les
```

```
[0]     validation_0-rmse:49613.34756   validation_1-rmse:48651.80067
[1000]  validation_0-rmse:44953.38978   validation_1-rmse:44047.26826
[2000]  validation_0-rmse:40741.42409   validation_1-rmse:39897.99180
[3000]  validation_0-rmse:36932.55249   validation_1-rmse:36152.96877
[4000]  validation_0-rmse:33488.78816   validation_1-rmse:32778.58905
[5000]  validation_0-rmse:30376.47126   validation_1-rmse:29724.64621
[6000]  validation_0-rmse:27564.90203   validation_1-rmse:26970.11200
[7000]  validation_0-rmse:25025.33002   validation_1-rmse:24487.22949
[8000]  validation_0-rmse:22732.36518   validation_1-rmse:22247.33411
[9000]  validation_0-rmse:20663.26886   validation_1-rmse:20232.71625
[10000] validation_0-rmse:18796.98303   validation_1-rmse:18420.93671
[11000] validation_0-rmse:17114.46885   validation_1-rmse:16793.97378
[12000] validation_0-rmse:15599.04543   validation_1-rmse:15337.44851
[13000] validation_0-rmse:14234.91545   validation_1-rmse:14034.02714
[14000] validation_0-rmse:13009.55132   validation_1-rmse:12861.20369
[15000] validation_0-rmse:11909.27151   validation_1-rmse:11814.03820
[16000] validation_0-rmse:10921.41923   validation_1-rmse:10885.17075
[17000] validation_0-rmse:10036.48617   validation_1-rmse:10060.36448
[18000] validation_0-rmse:9244.04939    validation_1-rmse:9329.03340
[19000] validation_0-rmse:8536.90128    validation_1-rmse:8692.28492
[20000] validation_0-rmse:7908.27851    validation_1-rmse:8136.33617
[21000] validation_0-rmse:7350.95349    validation_1-rmse:7650.90651
[22000] validation_0-rmse:6857.74191    validation_1-rmse:7232.41246
[23000] validation_0-rmse:6422.54608    validation_1-rmse:6872.46346
[24000] validation_0-rmse:6037.71189    validation_1-rmse:6565.05719
[25000] validation_0-rmse:5697.67239    validation_1-rmse:6303.42487
[26000] validation_0-rmse:5400.91612    validation_1-rmse:6083.98450
[27000] validation_0-rmse:5142.19342    validation_1-rmse:5889.97640
[28000] validation_0-rmse:4916.86745    validation_1-rmse:5730.18605
[29000] validation_0-rmse:4721.35113    validation_1-rmse:5597.52815
[30000] validation_0-rmse:4551.78140    validation_1-rmse:5487.15427
[31000] validation_0-rmse:4404.68723    validation_1-rmse:5397.06419
[32000] validation_0-rmse:4276.09631    validation_1-rmse:5323.11709
[33000] validation_0-rmse:4162.56153    validation_1-rmse:5260.81603
[34000] validation_0-rmse:4064.21189    validation_1-rmse:5209.26141
[35000] validation_0-rmse:3978.26744    validation_1-rmse:5167.67623
[36000] validation_0-rmse:3900.15336    validation_1-rmse:5137.28467
[37000] validation_0-rmse:3834.49331    validation_1-rmse:5113.26420
[38000] validation_0-rmse:3773.98609    validation_1-rmse:5093.56606
[39000] validation_0-rmse:3719.45922    validation_1-rmse:5076.64953
[40000] validation_0-rmse:3670.84787    validation_1-rmse:5064.54861
[41000] validation_0-rmse:3627.22205    validation_1-rmse:5056.63777
[42000] validation_0-rmse:3588.27238    validation_1-rmse:5051.17787
[43000] validation_0-rmse:3552.61811    validation_1-rmse:5047.88493
[44000] validation_0-rmse:3518.70662    validation_1-rmse:5043.98627
[45000] validation_0-rmse:3488.92719    validation_1-rmse:5040.58573
[45352] validation_0-rmse:3479.21194    validation_1-rmse:5040.90190
```

Out[27]:

```
▼                                XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_byt
ree=1,
             early_stopping_rounds=200, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='dep
thwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.0001, max_bin=256, max_cat_to_onehot=
4,
```

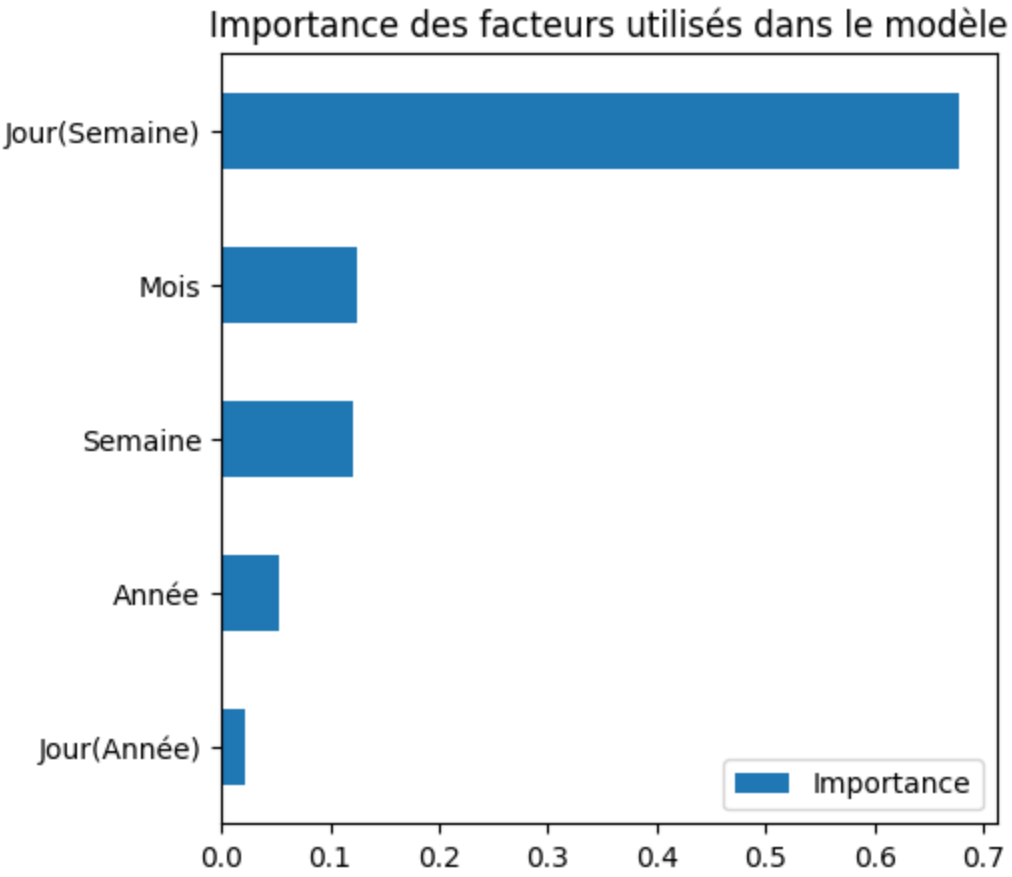
L'algorithme s'exécute de manière à faire diminuer l'erreur "root-mean-squared-error" ou "rmse" sur le set de données d'entraînement, ce qui la fait parrallèlement diminuer pour le set de test (les données que nous souhaitons prédire). L'algorithme entraîne le modèle jusqu'à ce que l'erreur calculée sur le set de test soit minimale, en effet celle-ci augmente quand le modèle apprend par coeur les données sur lesquelles il s'est entraîné (cela s'appelle l'overfitting et ce n'est pas souhaitable pour les prédictions sur des nouveaux sets de données).

In [28]:

```
imp = pd.DataFrame(
    data=reg.feature_importances_,
    index=reg.feature_names_in_,
    columns=['Importance'])

imp.sort_values('Importance').plot(
    figsize=(5,5),
    kind='barh',
    title='Importance des facteurs utilisés dans le modèle')
```

Out[28]:

```
<AxesSubplot: title={'center': 'Importance des facteurs utilisés dans le mo
dèle'}>
```



- SANS TEMPERATURE

In [29]:
```
ENTREES_TEMP = ['Jour(Semaine)', 'Semaine', 'Mois', 'Année', 'Jour(Année)',
SORTIES_TEMP = ['DAYTON_MW']
```

In [30]:
```
X_train_TEMP = train[ENTREES_TEMP]
Y_train_TEMP = train[SORTIES_TEMP]

X_test_TEMP = test[ENTREES_TEMP]
Y_test_TEMP = test[SORTIES_TEMP]
```

On entraîne le modèle de machine learning sur le set de données "**Train**" en utilisant la méthode de boosting de gradient extrême (XGBoost).

In [31]:
```
reg_TEMP = xgb.XGBRegressor(
    n_estimators=200000,         # Hauteur maximale de l'arbre de décision cr
    early_stopping_rounds=200,   # Nombre maximal de sorties de la fonction
    learning_rate=0.0001)        # Ratio d'apprentissage
reg_TEMP.fit(
    X_train_TEMP, Y_train_TEMP,
    eval_set=[(X_train_TEMP, Y_train_TEMP),(X_test_TEMP,Y_test_TEMP)],
    verbose=1000)               # Pas d'affichage des sorties (ici toutes les
```

```
[0]     validation_0-rmse:49613.32468   validation_1-rmse:48651.78319
[1000]  validation_0-rmse:44929.28977   validation_1-rmse:44014.56830
[2000]  validation_0-rmse:40691.33013   validation_1-rmse:39839.61243
[3000]  validation_0-rmse:36856.75346   validation_1-rmse:36093.42702
[4000]  validation_0-rmse:33386.92846   validation_1-rmse:32686.61389
[5000]  validation_0-rmse:30247.51032   validation_1-rmse:29605.80757
[6000]  validation_0-rmse:27407.24500   validation_1-rmse:26824.85592
[7000]  validation_0-rmse:24837.89766   validation_1-rmse:24319.18346
[8000]  validation_0-rmse:22513.49215   validation_1-rmse:22055.71959
[9000]  validation_0-rmse:20411.04270   validation_1-rmse:20013.96632
[10000] validation_0-rmse:18509.30237   validation_1-rmse:18170.37186
[11000] validation_0-rmse:16789.35400   validation_1-rmse:16504.28041
[12000] validation_0-rmse:15233.85175   validation_1-rmse:14995.29403
[13000] validation_0-rmse:13827.47567   validation_1-rmse:13633.03126
[14000] validation_0-rmse:12555.91535   validation_1-rmse:12402.13427
[15000] validation_0-rmse:11406.72240   validation_1-rmse:11296.11930
[16000] validation_0-rmse:10367.58659   validation_1-rmse:10299.70415
[17000] validation_0-rmse:9428.91987    validation_1-rmse:9403.54759
[18000] validation_0-rmse:8581.00618    validation_1-rmse:8592.74108
[19000] validation_0-rmse:7815.09595    validation_1-rmse:7863.80748
[20000] validation_0-rmse:7123.74114    validation_1-rmse:7207.71895
[21000] validation_0-rmse:6499.53443    validation_1-rmse:6616.24157
[22000] validation_0-rmse:5936.62597    validation_1-rmse:6081.52417
[23000] validation_0-rmse:5429.95851    validation_1-rmse:5604.17896
[24000] validation_0-rmse:4974.90682    validation_1-rmse:5178.01792
[25000] validation_0-rmse:4565.37073    validation_1-rmse:4796.16499
[26000] validation_0-rmse:4197.48279    validation_1-rmse:4455.30767
[27000] validation_0-rmse:3868.16462    validation_1-rmse:4150.43612
[28000] validation_0-rmse:3573.75031    validation_1-rmse:3878.07545
[29000] validation_0-rmse:3310.67383    validation_1-rmse:3635.42559
[30000] validation_0-rmse:3074.71243    validation_1-rmse:3419.40034
[31000] validation_0-rmse:2864.21371    validation_1-rmse:3225.58250
[32000] validation_0-rmse:2677.95011    validation_1-rmse:3055.61758
[33000] validation_0-rmse:2513.47847    validation_1-rmse:2906.67372
[34000] validation_0-rmse:2369.52704    validation_1-rmse:2778.96964
[35000] validation_0-rmse:2242.96376    validation_1-rmse:2665.34591
[36000] validation_0-rmse:2131.58792    validation_1-rmse:2563.21007
[37000] validation_0-rmse:2033.35308    validation_1-rmse:2473.40766
[38000] validation_0-rmse:1944.70054    validation_1-rmse:2390.67253
[39000] validation_0-rmse:1867.53464    validation_1-rmse:2317.01853
[40000] validation_0-rmse:1800.03234    validation_1-rmse:2252.78817
[41000] validation_0-rmse:1740.12817    validation_1-rmse:2195.19815
[42000] validation_0-rmse:1686.34574    validation_1-rmse:2146.33510
[43000] validation_0-rmse:1639.74450    validation_1-rmse:2104.77181
[44000] validation_0-rmse:1599.68227    validation_1-rmse:2069.11083
[45000] validation_0-rmse:1563.46979    validation_1-rmse:2035.73207
[46000] validation_0-rmse:1531.49221    validation_1-rmse:2007.94137
[47000] validation_0-rmse:1503.45059    validation_1-rmse:1983.59187
[48000] validation_0-rmse:1476.44965    validation_1-rmse:1959.81252
[49000] validation_0-rmse:1452.21122    validation_1-rmse:1935.05431
[50000] validation_0-rmse:1430.06628    validation_1-rmse:1914.82821
[51000] validation_0-rmse:1410.55441    validation_1-rmse:1895.61581
[52000] validation_0-rmse:1393.60293    validation_1-rmse:1879.44790
[53000] validation_0-rmse:1378.38246    validation_1-rmse:1866.24312
[54000] validation_0-rmse:1364.72960    validation_1-rmse:1855.03168
[55000] validation_0-rmse:1351.71522    validation_1-rmse:1843.97766
```

```
[56000] validation_0-rmse:1339.49434    validation_1-rmse:1834.06649
[57000] validation_0-rmse:1327.35486    validation_1-rmse:1824.81454
[58000] validation_0-rmse:1315.21062    validation_1-rmse:1816.53738
[59000] validation_0-rmse:1303.81911    validation_1-rmse:1808.68420
[60000] validation_0-rmse:1292.61388    validation_1-rmse:1801.35613
[61000] validation_0-rmse:1283.01692    validation_1-rmse:1794.64913
[62000] validation_0-rmse:1273.46646    validation_1-rmse:1788.43038
[63000] validation_0-rmse:1264.60754    validation_1-rmse:1784.72567
[64000] validation_0-rmse:1256.04189    validation_1-rmse:1780.11863
[65000] validation_0-rmse:1248.13232    validation_1-rmse:1775.99150
[66000] validation_0-rmse:1240.26672    validation_1-rmse:1772.93778
[67000] validation_0-rmse:1232.75620    validation_1-rmse:1769.08099
[68000] validation_0-rmse:1225.43581    validation_1-rmse:1765.15884
[69000] validation_0-rmse:1218.58639    validation_1-rmse:1760.84106
[70000] validation_0-rmse:1211.60148    validation_1-rmse:1756.07131
[71000] validation_0-rmse:1204.98827    validation_1-rmse:1751.75971
[72000] validation_0-rmse:1198.56428    validation_1-rmse:1748.13436
[73000] validation_0-rmse:1192.43065    validation_1-rmse:1744.91758
[74000] validation_0-rmse:1186.26307    validation_1-rmse:1741.26724
[75000] validation_0-rmse:1179.56882    validation_1-rmse:1738.36363
[76000] validation_0-rmse:1173.94371    validation_1-rmse:1735.53538
[77000] validation_0-rmse:1167.32377    validation_1-rmse:1732.66877
[78000] validation_0-rmse:1160.08524    validation_1-rmse:1729.27011
[79000] validation_0-rmse:1153.41471    validation_1-rmse:1725.97720
[80000] validation_0-rmse:1146.29999    validation_1-rmse:1723.56902
[81000] validation_0-rmse:1139.45247    validation_1-rmse:1721.11551
[82000] validation_0-rmse:1133.51520    validation_1-rmse:1718.91506
[83000] validation_0-rmse:1127.72163    validation_1-rmse:1716.33052
[84000] validation_0-rmse:1123.22808    validation_1-rmse:1714.90449
[85000] validation_0-rmse:1118.29393    validation_1-rmse:1712.93311
[86000] validation_0-rmse:1112.26103    validation_1-rmse:1711.14089
[87000] validation_0-rmse:1106.65697    validation_1-rmse:1709.41212
[88000] validation_0-rmse:1100.50411    validation_1-rmse:1707.10657
[89000] validation_0-rmse:1095.41078    validation_1-rmse:1704.66068
[90000] validation_0-rmse:1090.59086    validation_1-rmse:1702.22306
[91000] validation_0-rmse:1086.20246    validation_1-rmse:1700.88977
[92000] validation_0-rmse:1081.80162    validation_1-rmse:1699.55645
[93000] validation_0-rmse:1077.37246    validation_1-rmse:1698.94655
[94000] validation_0-rmse:1073.53292    validation_1-rmse:1697.72055
[95000] validation_0-rmse:1069.83062    validation_1-rmse:1696.62644
[96000] validation_0-rmse:1065.72932    validation_1-rmse:1695.58941
[97000] validation_0-rmse:1061.87948    validation_1-rmse:1695.17764
[97366] validation_0-rmse:1060.48040    validation_1-rmse:1695.10290
```

Out[31]:

```
▼                                XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_byt
ree=1,
             early_stopping_rounds=200, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='dep
thwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.0001, max_bin=256, max_cat_to_onehot=
4,
```

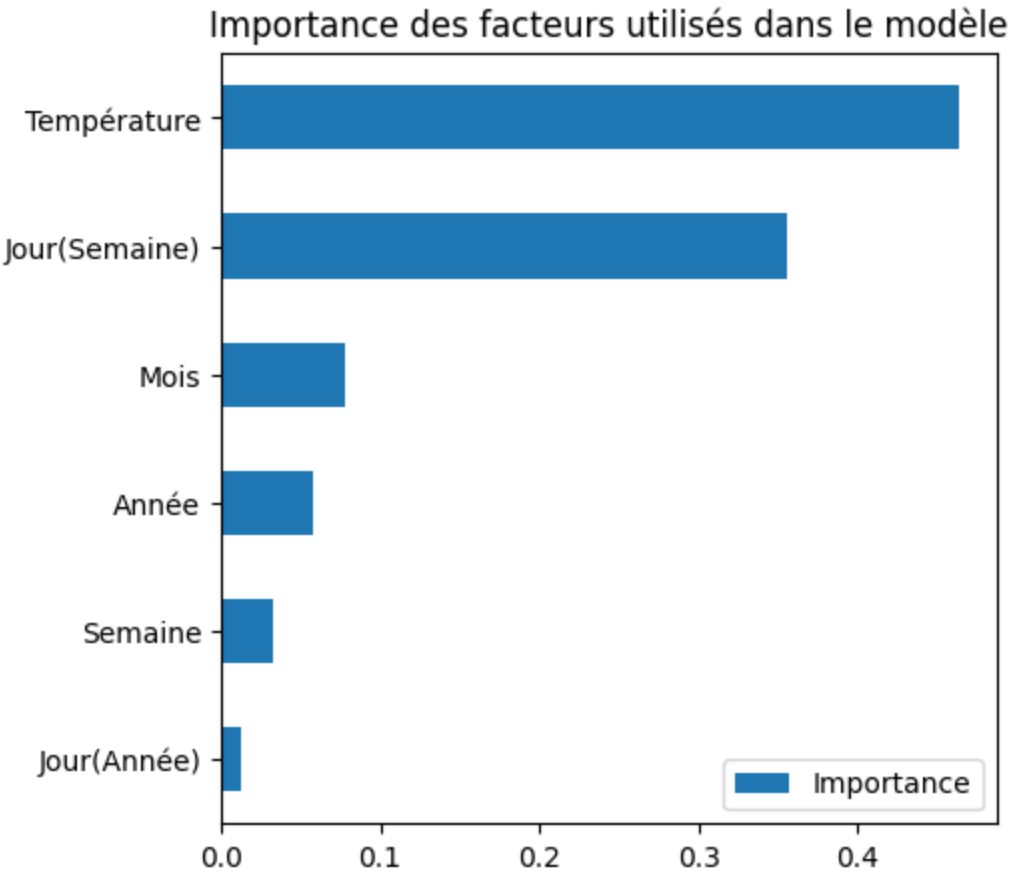
L'algorithme s'exécute de manière à faire diminuer l'erreur "root-mean-squared-error" ou "rmse" sur le set de données d'entraînement, ce qui la fait parrallèlement diminuer pour le set de test (les données que nous souhaitons prédire). L'algorithme entraîne le modèle jusqu'à ce que l'erreur calculée sur le set de test soit minimale, en effet celle-ci augmente quand le modèle apprend par coeur les données sur lesquelles il s'est entraîné (cela s'appelle l'overfitting et ce n'est pas souhaitable pour les prédictions sur des nouveaux sets de données).

In [32]:

```
imp_TEMP = pd.DataFrame(
    data=reg_TEMP.feature_importances_,
    index=reg_TEMP.feature_names_in_,
    columns=['Importance'])

imp_TEMP.sort_values('Importance').plot(
    figsize=(5,5),
    kind='barh',
    title='Importance des facteurs utilisés dans le modèle')
```

Out[32]:

```
<AxesSubplot: title={'center': 'Importance des facteurs utilisés dans le mo
dèle'}>
```



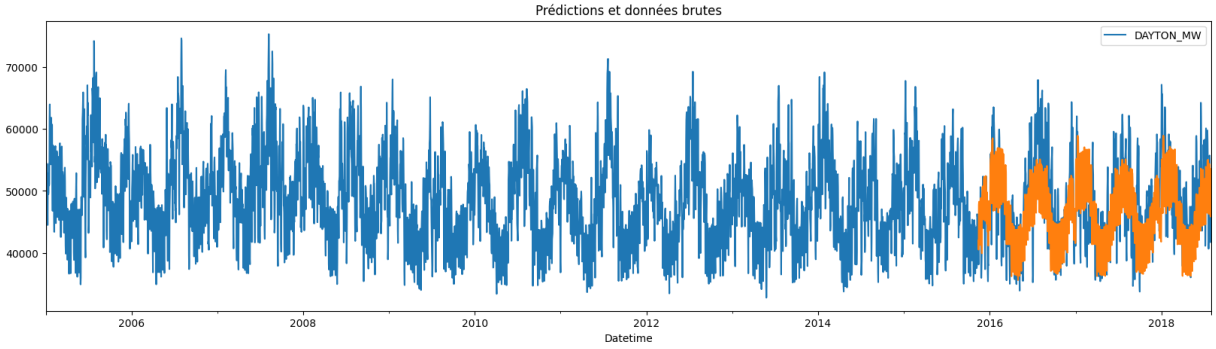
On trace les données présentes dans le set de données "**Test**" ainsi que les prédictions estimées par le modèle.

In [33]:
```
test['prediction'] = reg.predict(X_test)
dataset_final = dataset_final.merge(test[['prediction']], how='left', left_i
```

```
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\2034891001.py:1: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['prediction'] = reg.predict(X_test)
```

In [34]:
```
ax = dataset_final[['DAYTON_MW']].plot(figsize=(20,5))
dataset_final['prediction'].plot(ax=ax)
ax.set_title('Prédictions et données brutes')
```

Out[34]:
```
Text(0.5, 1.0, 'Prédictions et données brutes')
```

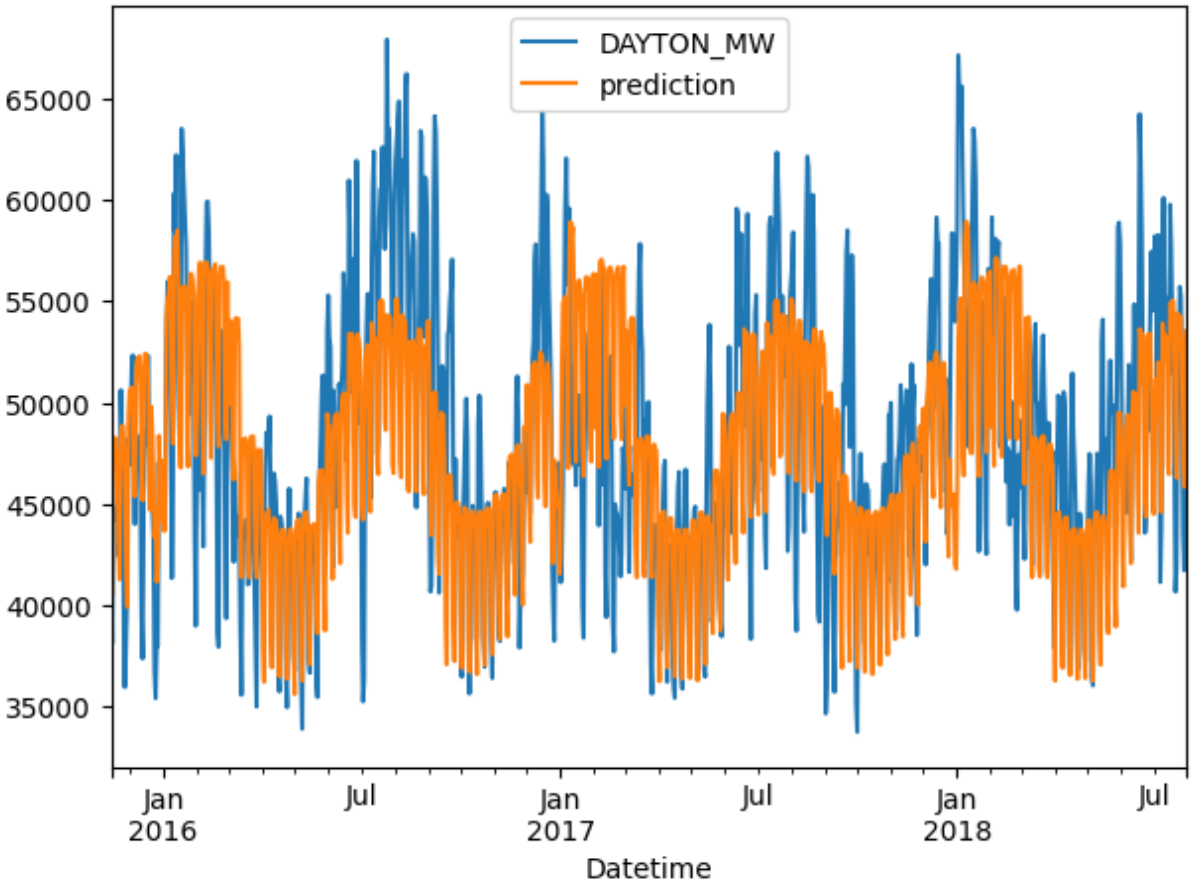


Intégralité des données prédites.

In [35]:
```
test_predic = dataset_final.iloc[int(nb_lines*0.8)+1:]
```

In [36]:
```
test_predic.plot(y=['DAYTON_MW','prediction'],figsize=(7,5))
```

Out[36]: <AxesSubplot: xlabel='Datetime'>

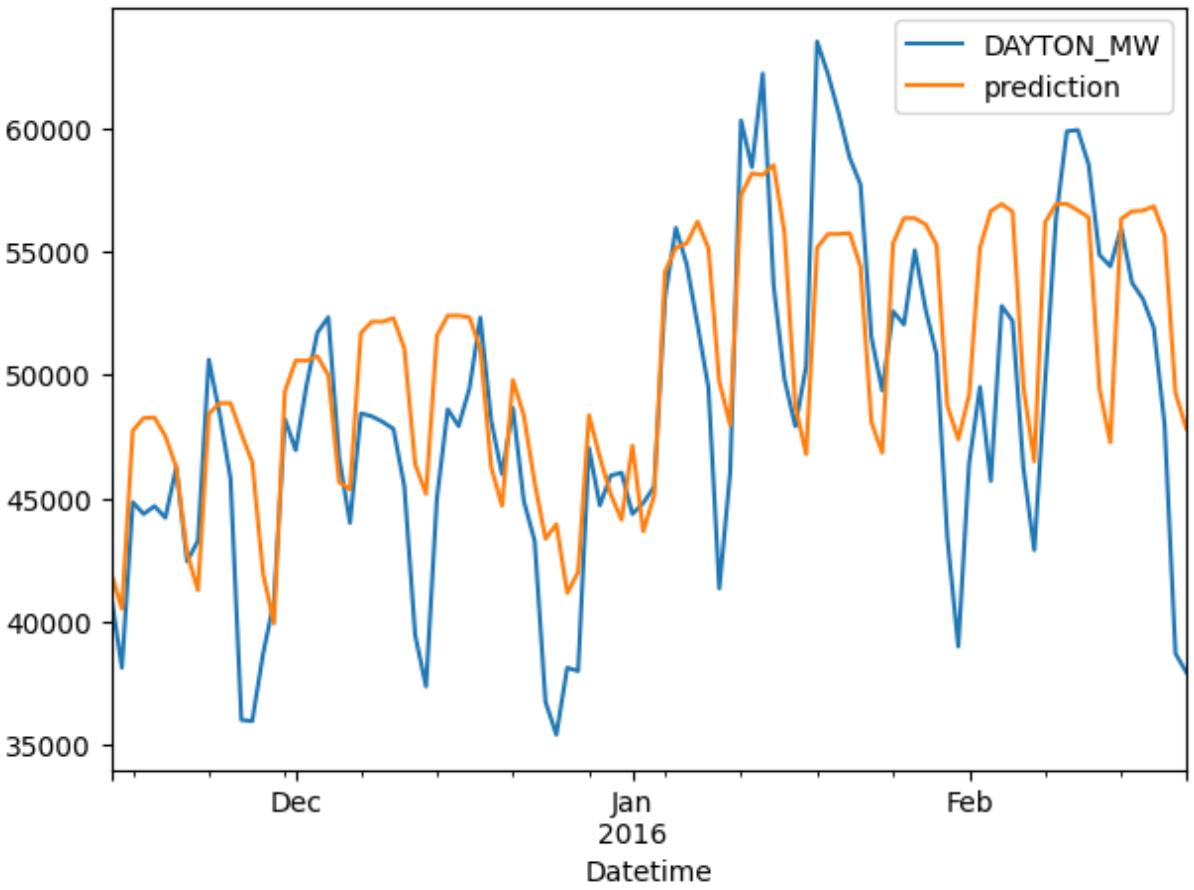


Centaine de données prédites.

In [37]:
```
test_predic = dataset_final.iloc[int(nb_lines*0.8)+1:int(nb_lines*0.8)+101]
```

In [38]:
```
test_predic.plot(y=['DAYTON_MW','prediction'],figsize=(7,5))
```

Out[38]: <AxesSubplot: xlabel='Datetime'>



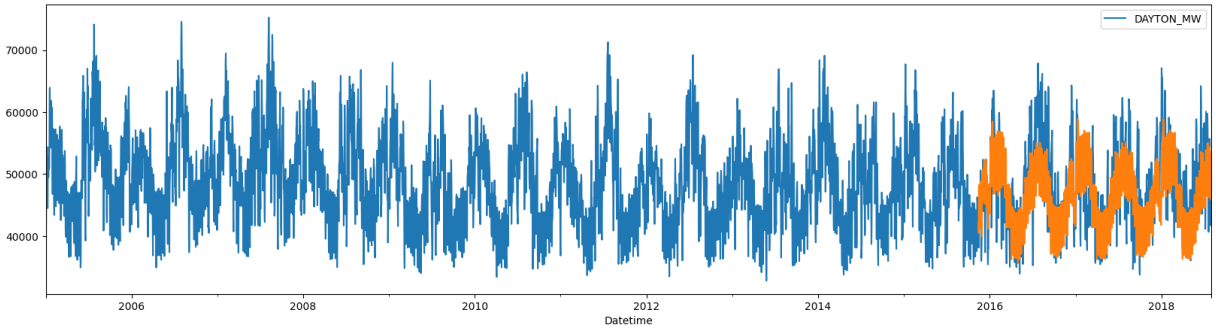
On trace les données présentes dans le set de données "**Test**" ainsi que les prédictions estimées par le modèle.

In [39]:
```
test['prediction avec temperature'] = reg_TEMP.predict(X_test_TEMP)
dataset_final = dataset_final.merge(test[['prediction avec temperature']], h
```

```
C:\Users\tautu\AppData\Local\Temp\ipykernel_12908\631696765.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  test['prediction avec temperature'] = reg_TEMP.predict(X_test_TEMP)
```

In [40]:
```
ax_temp = dataset_final[['DAYTON_MW']].plot(figsize=(20,5))
dataset_final['prediction'].plot(ax=ax_temp)
ax.set_title('Prédictions et données brutes')
```

Out[40]: Text(0.5, 1.0, 'Prédictions et données brutes')

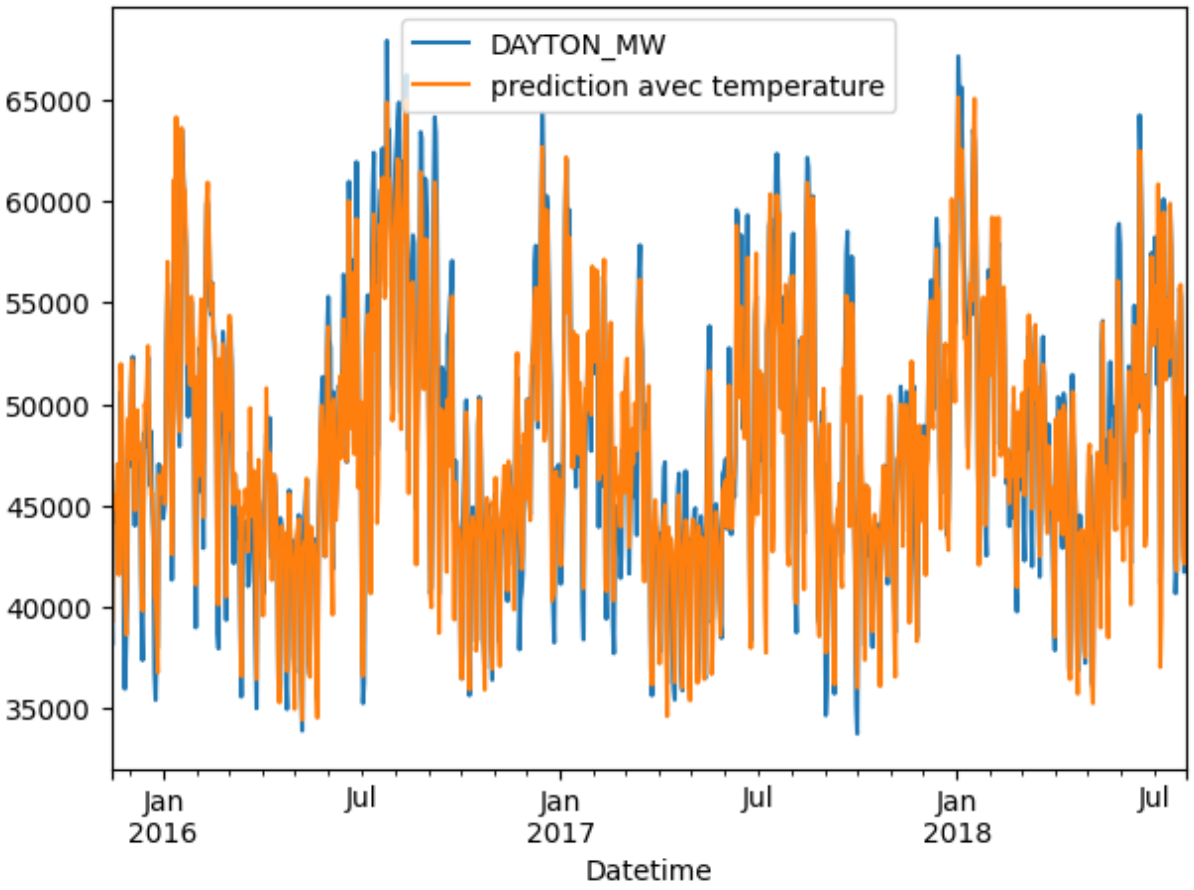


Intégralité des données prédites.

In [41]:
```
test_predic_temp = dataset_final.iloc[int(nb_lines*0.8)+1:]
```

In [42]:
```
test_predic_temp.plot(y=['DAYTON_MW','prediction avec temperature'],figsize=
```

Out[42]: <AxesSubplot: xlabel='Datetime'>

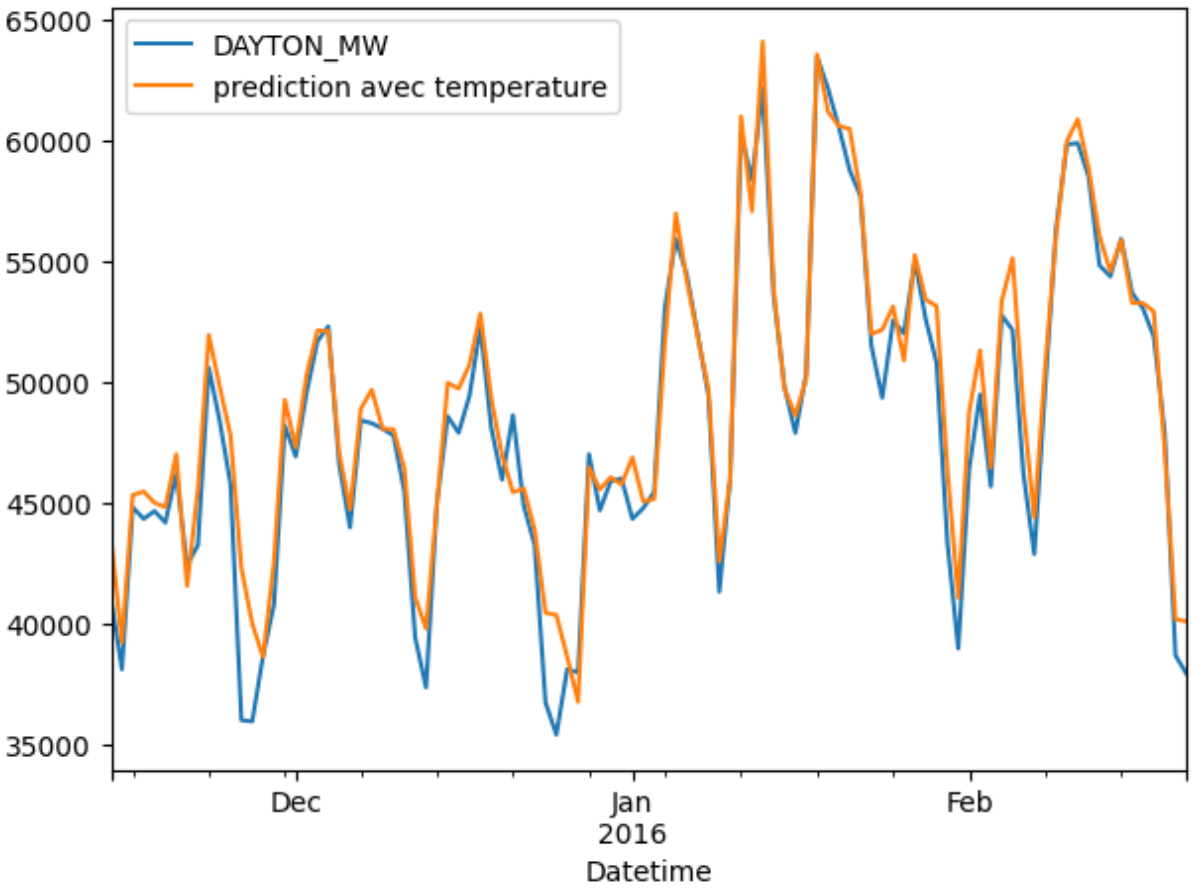


Centaine de données prédites.

In [43]:
```
test_predic_temp = dataset_final.iloc[int(nb_lines*0.8)+1:int(nb_lines*0.8)+
```

In [48]:
```
test_predic_temp.plot(y=['DAYTON_MW','prediction avec temperature'],figsize=
```

Out[48]: <AxesSubplot: xlabel='Datetime'>



Calcul de l'erreur (root_mean_squared_error) entre les prédictions et les données réelles brutes du set "**Test**".

In [ ]:
```
score = mean_squared_error(test['DAYTON_MW'], test['prediction'], squared=Fa
print('RMSE =', score)
```

```
RMSE = 5040.5360585249755
```

In [ ]:
```
score_TEMP = mean_squared_error(test['DAYTON_MW'], test['prediction avec tem
print('RMSE =', score_TEMP)
```

```
RMSE = 1695.0614233162114
```