C_b_1 : Recherche de modèles

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import locale
import calendar
import holidays
from rich import print
from datetime import date
from references import *
from src import *

import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse
from sklearn.model_selection import TimeSeriesSplit

import warnings

warnings.filterwarnings("ignore")
plt.style.use("fivethirtyeight")
pd.options.mode.chained_assignment = None
pd.set_option("display.max_columns", 500)
pd.set_option("display.width", 1000)

locale.setlocale(locale.LC_ALL, "fr_CA.UTF-8")
```

Out[1]:
```
'fr_CA.UTF-8'
```

In [2]:
```
print("Version XGBoost :", xgb.__version__)
```

```
Version XGBoost : 1.7.6
```

In [3]:
```
# %load_ext jupyter_black

import black
import jupyter_black

jupyter_black.load(
    lab=True,
    line_length=55,
    target_version=black.TargetVersion.PY311,
)
```

XGBoost

https://en.wikipedia.org/wiki/XGBoost

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed **gradient-boosted decision tree (GBDT)** machine learning library. It provides **parallel tree boosting** and is the leading machine learning library for regression, classification, and ranking problems.

> XGBoost gained significant favor in the last few years as a result of helping individuals and teams win virtually every Kaggle structured data competition. In these competitions, companies and researchers post data after which statisticians and data miners compete to produce the best models for predicting and describing the data. (https://www.nvidia.com/en-us/glossary/data-science/xgboost/)

Exemple inspirant ce sujet

Vidéo partie 1 : https://youtu.be/vV12dGe_Fho?si=Zifv1O512VNiqJ8W

Première partie - Création d'un modèle initial

Import des données et création des features

Nous utilisons les features sans valeurs catégoriques, car les catérories (Été, Printemps, etc..) causent des problèmes au modèle plus loin.

Nous enlevons aussi les valeurs NA.

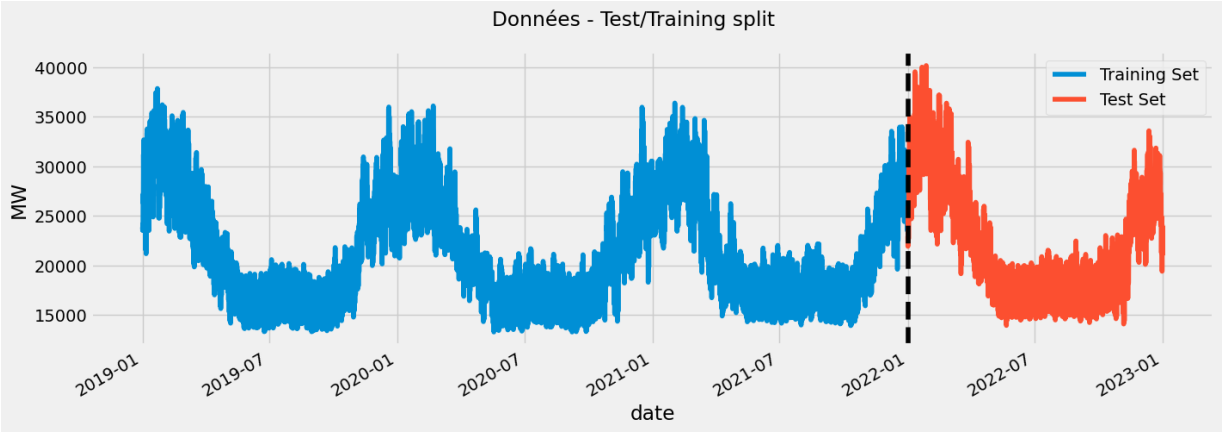
In [4]:
```
df = import_and_create_features_no_categorical(
    fin="20221231"
).dropna()
```

Séparation des données d'entraînement et de test

Séparons les données en ensemble d'entraînement et ensemble de test, où la dernière année complète (2022) servire de test.

In [5]:
```
date_slit = "2022-01-01"

train = df.iloc[df.index < date_slit]
test = df.iloc[df.index >= date_slit]

fig, ax = plt.subplots(figsize=(15, 5))
train["MW"].plot(
```

```
    ax=ax, label="Training set", ylabel="MW"
)
test["MW"].plot(ax=ax, label="Test set")
ax.legend(["Training Set", "Test Set"])
ax.axvline(date_slit, color="black", ls="--")
fig.suptitle("Données - Test/Training split")
plt.show()
```



In [6]:
```
df.head()
```

Out[6]:

date	MW	Temp	hourofday	quarter	year	dayofyear	dayofmonth	weekof
2019-01-01 01:00:00	23762.55	0.0	1	1	2019	1	1	
2019-01-01 02:00:00	23830.23	-0.2	2	1	2019	1	1	
2019-01-01 03:00:00	23608.07	-0.5	3	1	2019	1	1	
2019-01-01 04:00:00	23562.48	-1.0	4	1	2019	1	1	
2019-01-01 05:00:00	23546.16	-1.1	5	1	2019	1	1	

Création du modèle

In [7]:
```
df.columns
```

Out[7]:
```
Index(['MW', 'Temp', 'hourofday', 'quarter', 'year', 'dayofyear', 'dayofmon
th', 'weekofyear', 'month', 'dayofweek', 'season', 'isWeekend', 'isHolida
y', 'day_sin', 'day_cos', 'year_sin', 'year_cos', 'CDD_21', 'HDD_18', 'CDD_
24', 'HDD_16', 'DT_18-21', 'DT_16-24', 'DT_18', 'DT_21', 'Temp_LAG_t-1h',
'DT_18-21_LAG_t-1h', 'DT_16-24_LAG_t-1h', 'DT_18_LAG_t-1h', 'DT_21_LAG_t-1
h', 'Temp_LAG_t-2h', 'DT_18-21_LAG_t-2h', 'DT_16-24_LAG_t-2h', 'DT_18_LAG_t
-2h', 'DT_21_LAG_t-2h', 'Temp_LAG_t-3h', 'DT_18-21_LAG_t-3h', 'DT_16-24_LAG
_t-3h', 'DT_18_LAG_t-3h', 'DT_21_LAG_t-3h', 'Temp_LAG_t-4h', 'DT_18-21_LAG_
t-4h', 'DT_16-24_LAG_t-4h', 'DT_18_LAG_t-4h', 'DT_21_LAG_t-4h', 'Temp_LAG_t
-6h', 'DT_18-21_LAG_t-6h', 'DT_16-24_LAG_t-6h', 'DT_18_LAG_t-6h', 'DT_21_LA
G_t-6h', 'Temp_LAG_t-24h', 'DT_18-21_LAG_t-24h', 'DT_16-24_LAG_t-24h', 'DT_
18_LAG_t-24h', 'DT_21_LAG_t-24h', 'Temp_MOYMOBILE_t-1h', 'DT_18-21_MOYMOBIL
E_t-1h', 'DT_16-24_MOYMOBILE_t-1h', 'DT_18_MOYMOBILE_t-1h', 'DT_21_MOYMOBIL
E_t-1h',
       'Temp_MOYMOBILE_t-2h', 'DT_18-21_MOYMOBILE_t-2h', 'DT_16-24_MOYMOBIL
E_t-2h', 'DT_18_MOYMOBILE_t-2h', 'DT_21_MOYMOBILE_t-2h', 'Temp_MOYMOBILE_t-
3h', 'DT_18-21_MOYMOBILE_t-3h', 'DT_16-24_MOYMOBILE_t-3h', 'DT_18_MOYMOBILE
_t-3h', 'DT_21_MOYMOBILE_t-3h', 'Temp_MOYMOBILE_t-4h', 'DT_18-21_MOYMOBILE_
t-4h', 'DT_16-24_MOYMOBILE_t-4h', 'DT_18_MOYMOBILE_t-4h', 'DT_21_MOYMOBILE_
t-4h', 'Temp_MOYMOBILE_t-6h', 'DT_18-21_MOYMOBILE_t-6h', 'DT_16-24_MOYMOBIL
E_t-6h', 'DT_18_MOYMOBILE_t-6h', 'DT_21_MOYMOBILE_t-6h', 'Temp_MOYMOBILE_t-
24h', 'DT_18-21_MOYMOBILE_t-24h', 'DT_16-24_MOYMOBILE_t-24h', 'DT_18_MOYMOB
ILE_t-24h', 'DT_21_MOYMOBILE_t-24h'],
      dtype='object')
```

Sélection de certaines features seulement pour débuter avec un modèle simple.

In [8]:
```
FEATURES_PRELIM = [
    "Temp",
    "hourofday",
    "dayofyear",
    "weekofyear",
    "season",
    "month",
    "DT_18",
]
TARGET = "MW"

X_train = train[FEATURES_PRELIM]
y_train = train[TARGET]

X_test = test[FEATURES_PRELIM]
y_test = test[TARGET]
```

In [9]:
```
n_estim = 2000  # nb trees

reg = xgb.XGBRegressor(
    n_estimators=n_estim,
    early_stopping_rounds=50,
    learning_rate=0.01,
)

reg.fit(
    X_train,
    y_train,
```

```
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=100,
)
```

```
[0]     validation_0-rmse:21674.01427   validation_1-rmse:22647.83498
[100]   validation_0-rmse:8034.79466    validation_1-rmse:8999.61141
[200]   validation_0-rmse:3125.22792    validation_1-rmse:4069.04895
[300]   validation_0-rmse:1495.89873    validation_1-rmse:2368.65531
[400]   validation_0-rmse:1057.38956    validation_1-rmse:1836.10044
[500]   validation_0-rmse:954.31266     validation_1-rmse:1660.22385
[600]   validation_0-rmse:923.92462     validation_1-rmse:1598.61130
[700]   validation_0-rmse:907.35117     validation_1-rmse:1579.94158
[800]   validation_0-rmse:891.96170     validation_1-rmse:1576.19202
[874]   validation_0-rmse:881.87242     validation_1-rmse:1577.22657
```

Out[9]:

▾ XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_ty
pes=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_
type=None,
             interaction_constraints=None, learning_rate=0.01, max_
bin=None,
```

In [10]:

```
rmse_initial = reg.best_score

print(
    f"Le meilleur RMSE sur les données de tests est : {rmse_initial:0.2f}"
)
```

```
Le meilleur RMSE sur les données de tests est : 1576.00
```

Pour référence, nous avions un RMSE de **2399** sur la régression simple (Notebook B_c_3_Test_Statistique).

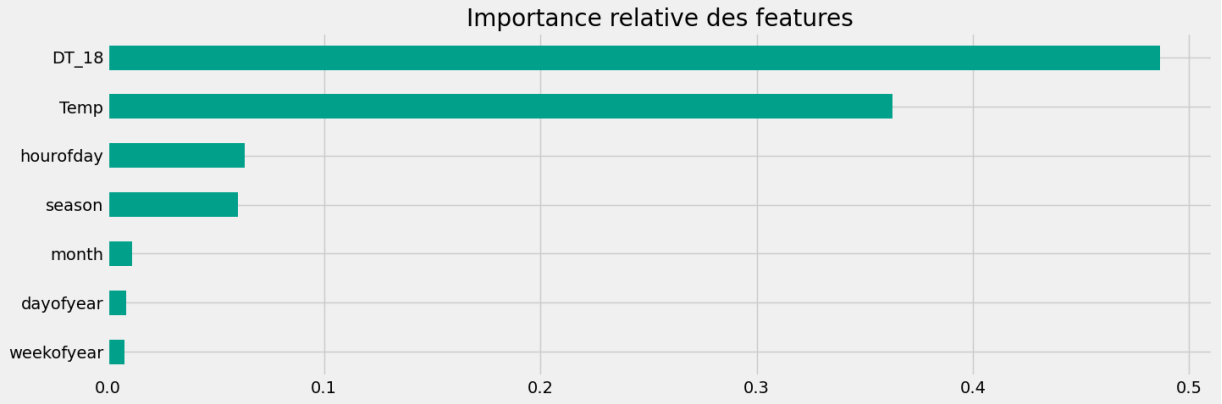
Importance des Features

Importance relative des différents features.

In [11]:

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

fi.sort_values("Importance").plot(
    figsize=(15, 5),
    kind="barh",
    title="Importance relative des features",
```

```
    legend=False,
    color=colors_pal[1],
)
plt.show()
```



In [12]:
```
fi.sort_values("Importance", ascending=False)
```

Out[12]:

	Importance
DT_18	0.486549
Temp	0.362852
hourofday	0.063172
season	0.060199
month	0.011106
dayofyear	0.008453
weekofyear	0.007667

In [13]:
```
ratio = (
    fi.loc[fi.index == "DT_18"]["Importance"][0]
    / fi.loc[fi.index == "Temp"]["Importance"][0]
)
print(
    f"Le ratio d'importance entre le DeltaT et la température en tant que te
)
```

```
Le ratio d'importance entre le DeltaT et la température en tant que te
importance relative, donc son degré de potentiel de prévision un par r
```

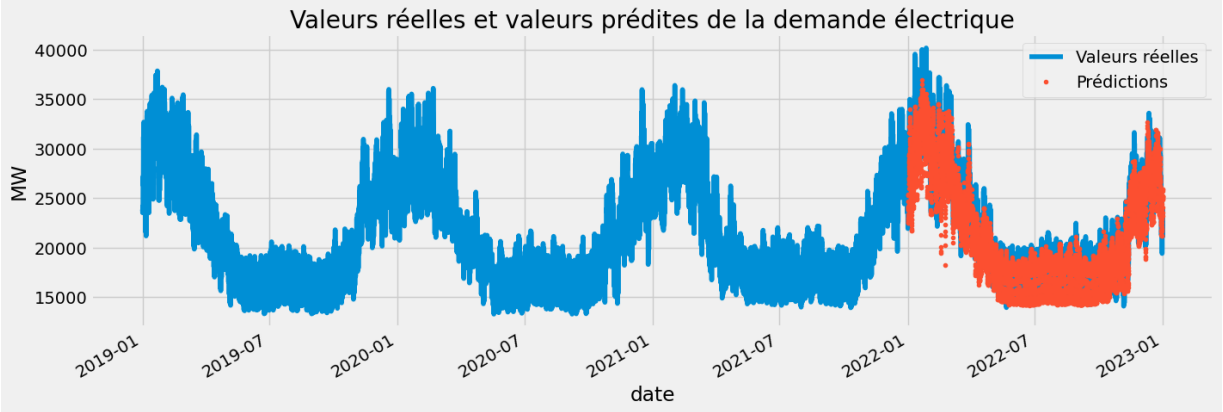
Aussi, les variables de température sont beaucoup plus importantes que les autres variables basées sur les temps et dates, ce que nous pouvions supposer avec l'évaluation statistique précédente.

Prédictions

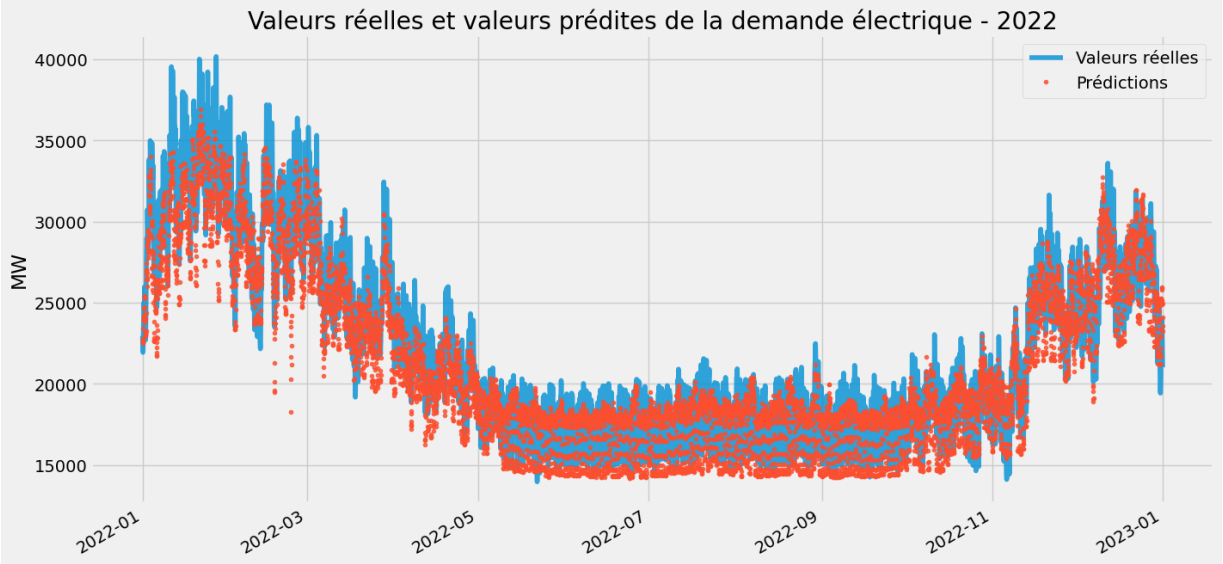
In [14]:
```
test["prediction"] = reg.predict(X_test)

df_pred = df.merge(
```

```
    test[["prediction"]],
    how="left",
    left_index=True,
    right_index=True,
)
```

In [15]:
```
ax = df_pred.MW.plot(figsize=(15, 5), ylabel="MW")
df_pred.prediction.plot(ax=ax, style=".")
plt.legend(["Valeurs réelles", "Prédictions"])
ax.set_title(
    "Valeurs réelles et valeurs prédites de la demande électrique"
)

plt.show()
```

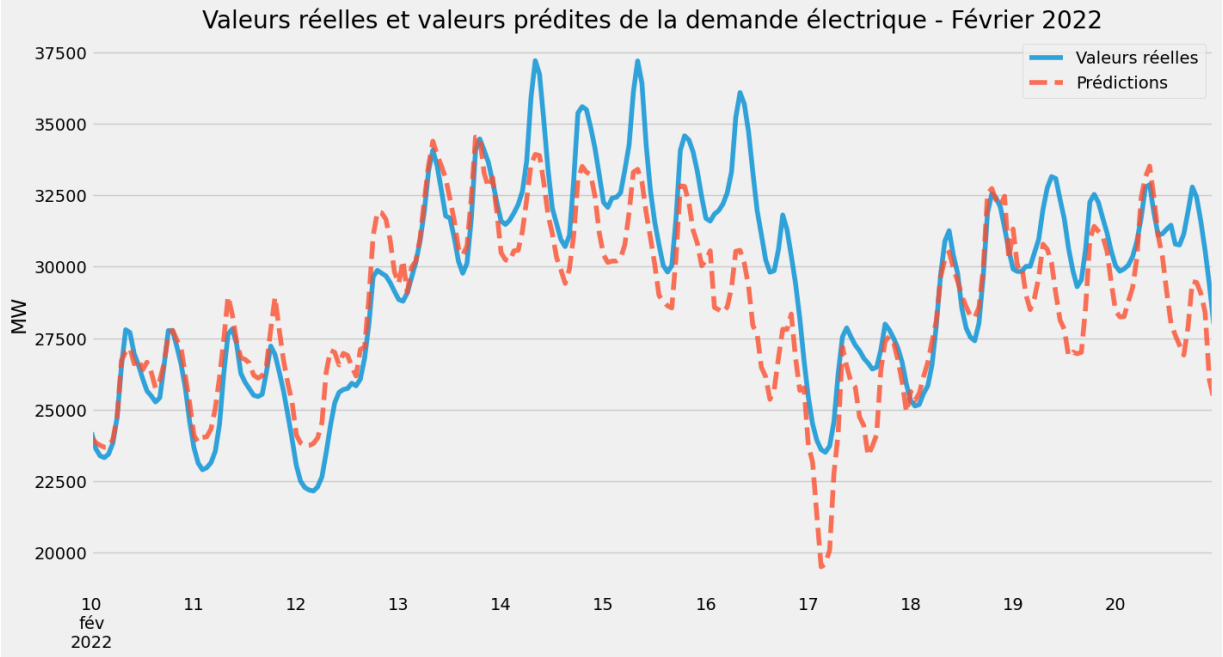


In [16]:
```
df_2022 = df_pred["20220101":"20221231"]

ax = df_2022.MW.plot(
    figsize=(15, 8), ylabel="MW", alpha=0.8
)
df_2022.prediction.plot(
    ax=ax, style=".", alpha=0.8, xlabel=""
)
plt.legend(["Valeurs réelles", "Prédictions"])

ax.set_title(
    "Valeurs réelles et valeurs prédites de la demande électrique - 2022"
)

plt.show()
```



Nous pouvons voir que les journées extrêmes semblent être sous-estimées, par exemple en février 2022.

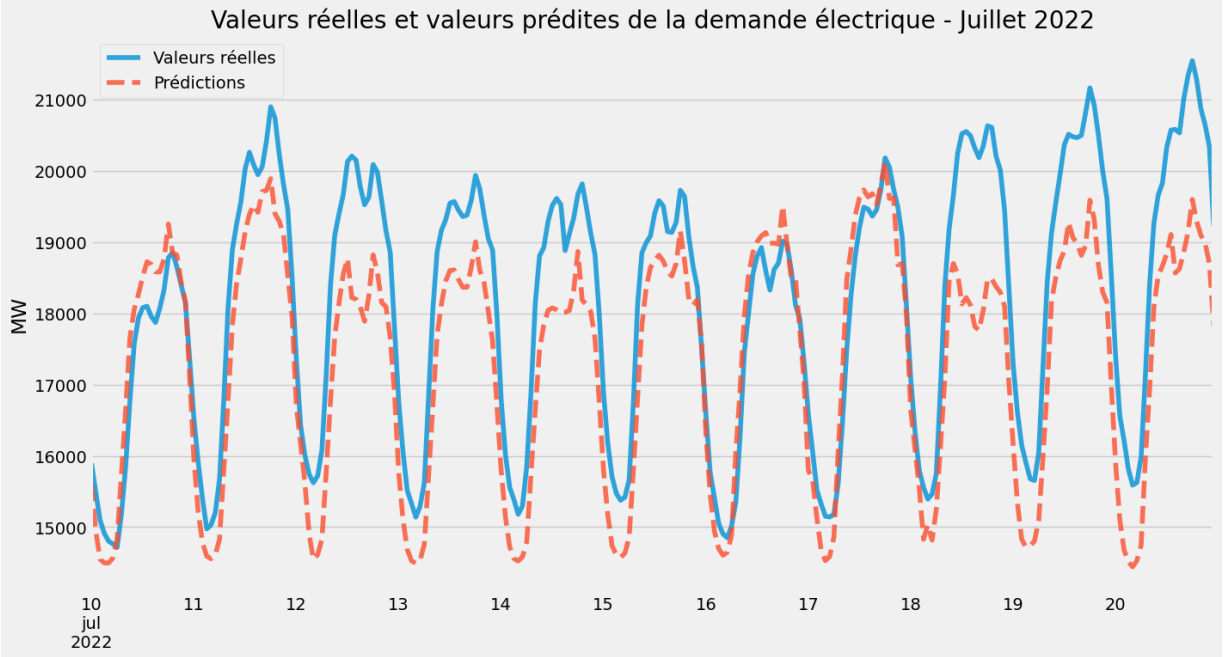
In [17]:
```
df_fev_2022 = df_pred["20220210":"20220220"]

ax = df_fev_2022.MW.plot(
    figsize=(15, 8), ylabel="MW", alpha=0.8
)
df_fev_2022.prediction.plot(
    ax=ax, style="--", alpha=0.8, xlabel=""
)
plt.legend(["Valeurs réelles", "Prédictions"])

ax.set_title(
    "Valeurs réelles et valeurs prédites de la demande électrique – Février
)

plt.show()
```



Nous pouvons clairement voir les tendances des prévisions des valeurs réelles, copiant même la double pointe journalière le matin et en fin de journée.

Lors des journées les plus extrêmes par contre, le modèle ne prédit pas de valeurs au-delà de 35000 MW, alors que la réalité va à près de 37500 MW. Il n'y a possiblement pas assez de données dans les ensembles d'entraînement qui se rendent à ce niveau pour qu'il apprenne la limite.

In [18]:
```
df_juin_2022 = df_pred["20220710":"20220720"]

ax = df_juin_2022.MW.plot(
    figsize=(15, 8), ylabel="MW", alpha=0.8
)
df_juin_2022.prediction.plot(
    ax=ax, style="--", alpha=0.8, xlabel=""
)
plt.legend(["Valeurs réelles", "Prédictions"])

ax.set_title(
    "Valeurs réelles et valeurs prédites de la demande électrique - Juillet
)

plt.show()
```



En été, les prédictions semblent sous-estimer la demande en creux et en sommet de la demande, même si la forme est très bien prédite (la double pointe).

Évaluation des métriques

In [19]:
```
# RMSE
score_RMSE = np.sqrt(
    mean_squared_error(test.MW, test.prediction)
)
print(f"RMSE : {score_RMSE:0.2f}")
```

```
RMSE : 1576.00
```

In [20]:
```
# Erreurs : erreur absolue moyenne sur la journée
test["error"] = np.abs(
    test[TARGET] - test["prediction"]
)
```

In [21]:
```
test["jour"] = test.index.date
```

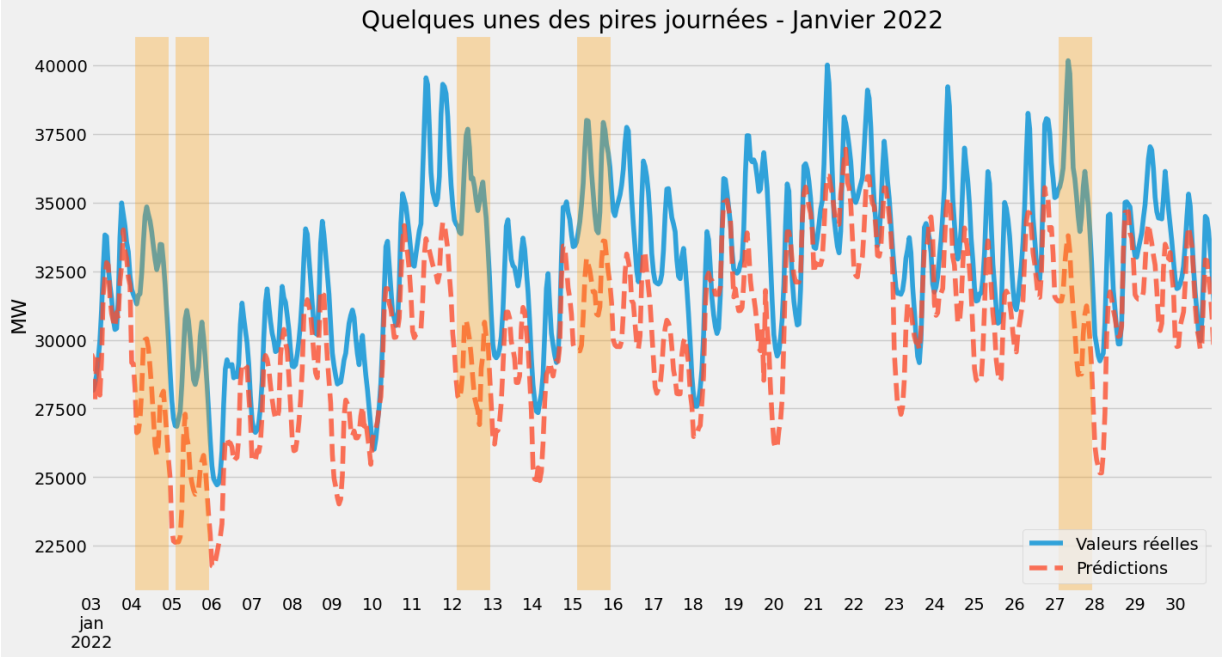
In [22]:
```
pire_journee = (
    test.groupby("jour")["error"]
    .mean()
    .sort_values(ascending=False)
    .head(5)
)
pire_journee
```

Out[22]:
```
jour
2022-01-12    5911.056104
2022-01-27    4903.238649
2022-01-04    4811.152220
2022-01-05    4268.765098
2022-01-15    4116.055391
Name: error, dtype: float64
```

Les pires journées de prédiction sont durant le mois de janvier, quand les températures sont froides et que les pointes ne sont pas bien prédites par le modèle.

Visualisons ces journées.

In [23]:
```
df_janv_2022 = df_pred["20220103":"20220130"]

ax = df_janv_2022.MW.plot(
    figsize=(15, 8), ylabel="MW", alpha=0.8
)
df_janv_2022.prediction.plot(
    ax=ax, style="--", alpha=0.8, xlabel=""
)
plt.legend(["Valeurs réelles", "Prédictions"])

ax.set_title(
    "Quelques unes des pires journées - Janvier 2022"
)

for jour in pire_journee.index.to_list():
    ax.axvline(
        datetime(
            jour.year, jour.month, jour.day, 12, 0
        ),
        color=colors_pal[11],
        linewidth=28,
        alpha=0.3,
    )

plt.show()
```



In [24]:
```
meilleure_journee = (
    test.groupby("jour")["error"]
    .mean()
    .sort_values(ascending=True)
    .head(5)
)
meilleure_journee
```

Out[24]:
```
jour
2022-05-01    235.759971
2022-06-25    254.084676
2022-10-16    270.456375
2022-10-26    289.359010
2022-09-25    299.622476
Name: error, dtype: float64
```

À première vue, les meilleures journées de prédiction semblent être en mi-saison (printemps, début été et automne).

2e passe : utilisation de l'ensemble des features

Tentons de trouver les meilleurs *features* qui sont déterminants pour le modèle.

In [25]:
```
FEATURES_TOUS = [
    "Temp",
    "hourofday",
    "quarter",
    "year",
    "dayofyear",
    "dayofmonth",
    "weekofyear",
    "month",
    "dayofweek",
```

```
    "season",
    "isWeekend",
    "isHoliday",
    "day_sin",
    "day_cos",
    "year_sin",
    "year_cos",
    "CDD_21",
    "HDD_18",
    "CDD_24",
    "HDD_16",
    "DT_18-21",
    "DT_16-24",
    "DT_18",
    "DT_21",
    "Temp_LAG_t-1h",
    "DT_18-21_LAG_t-1h",
    "DT_16-24_LAG_t-1h",
    "DT_18_LAG_t-1h",
    "DT_21_LAG_t-1h",
    "Temp_LAG_t-2h",
    "DT_18-21_LAG_t-2h",
    "DT_16-24_LAG_t-2h",
    "DT_18_LAG_t-2h",
    "DT_21_LAG_t-2h",
    "Temp_LAG_t-3h",
    "DT_18-21_LAG_t-3h",
    "DT_16-24_LAG_t-3h",
    "DT_18_LAG_t-3h",
    "DT_21_LAG_t-3h",
    "Temp_LAG_t-4h",
    "DT_18-21_LAG_t-4h",
    "DT_16-24_LAG_t-4h",
    "DT_18_LAG_t-4h",
    "DT_21_LAG_t-4h",
    "Temp_LAG_t-6h",
    "DT_18-21_LAG_t-6h",
    "DT_16-24_LAG_t-6h",
    "DT_18_LAG_t-6h",
    "DT_21_LAG_t-6h",
    "Temp_LAG_t-24h",
    "DT_18-21_LAG_t-24h",
    "DT_16-24_LAG_t-24h",
    "DT_18_LAG_t-24h",
    "DT_21_LAG_t-24h",
    "Temp_MOYMOBILE_t-1h",
    "DT_18-21_MOYMOBILE_t-1h",
    "DT_16-24_MOYMOBILE_t-1h",
    "DT_18_MOYMOBILE_t-1h",
    "DT_21_MOYMOBILE_t-1h",
    "Temp_MOYMOBILE_t-2h",
    "DT_18-21_MOYMOBILE_t-2h",
    "DT_16-24_MOYMOBILE_t-2h",
    "DT_18_MOYMOBILE_t-2h",
    "DT_21_MOYMOBILE_t-2h",
    "Temp_MOYMOBILE_t-3h",
```

```
    "DT_18-21_MOYMOBILE_t-3h",
    "DT_16-24_MOYMOBILE_t-3h",
    "DT_18_MOYMOBILE_t-3h",
    "DT_21_MOYMOBILE_t-3h",
    "Temp_MOYMOBILE_t-4h",
    "DT_18-21_MOYMOBILE_t-4h",
    "DT_16-24_MOYMOBILE_t-4h",
    "DT_18_MOYMOBILE_t-4h",
    "DT_21_MOYMOBILE_t-4h",
    "Temp_MOYMOBILE_t-6h",
    "DT_18-21_MOYMOBILE_t-6h",
    "DT_16-24_MOYMOBILE_t-6h",
    "DT_18_MOYMOBILE_t-6h",
    "DT_21_MOYMOBILE_t-6h",
    "Temp_MOYMOBILE_t-24h",
    "DT_18-21_MOYMOBILE_t-24h",
    "DT_16-24_MOYMOBILE_t-24h",
    "DT_18_MOYMOBILE_t-24h",
    "DT_21_MOYMOBILE_t-24h",
]

X_train = train[FEATURES_TOUS]
y_train = train[TARGET]

X_test = test[FEATURES_TOUS]
y_test = test[TARGET]
```

In [26]:

```
n_estim = 2000  # nb trees

reg = xgb.XGBRegressor(
    n_estimators=n_estim,
    early_stopping_rounds=50,
    learning_rate=0.01,
)

reg.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=100,
)
```

```
[0]     validation_0-rmse:21673.83985   validation_1-rmse:22647.24468
[100]   validation_0-rmse:8010.26111    validation_1-rmse:8916.87065
[200]   validation_0-rmse:3042.98159    validation_1-rmse:3903.88951
[300]   validation_0-rmse:1294.07597    validation_1-rmse:2093.17041
[400]   validation_0-rmse:745.14084     validation_1-rmse:1464.08853
[500]   validation_0-rmse:592.63943     validation_1-rmse:1242.50774
[600]   validation_0-rmse:538.43881     validation_1-rmse:1167.62203
[700]   validation_0-rmse:511.27227     validation_1-rmse:1144.82015
[800]   validation_0-rmse:493.57035     validation_1-rmse:1136.57819
[900]   validation_0-rmse:477.79395     validation_1-rmse:1132.42706
[1000]  validation_0-rmse:465.04468     validation_1-rmse:1130.84258
[1100]  validation_0-rmse:453.13285     validation_1-rmse:1130.27342
[1162]  validation_0-rmse:446.33275     validation_1-rmse:1130.17399
```

Out[26]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_ty
pes=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_
type=None,
             interaction_constraints=None, learning_rate=0.01, max_
bin=None,
```

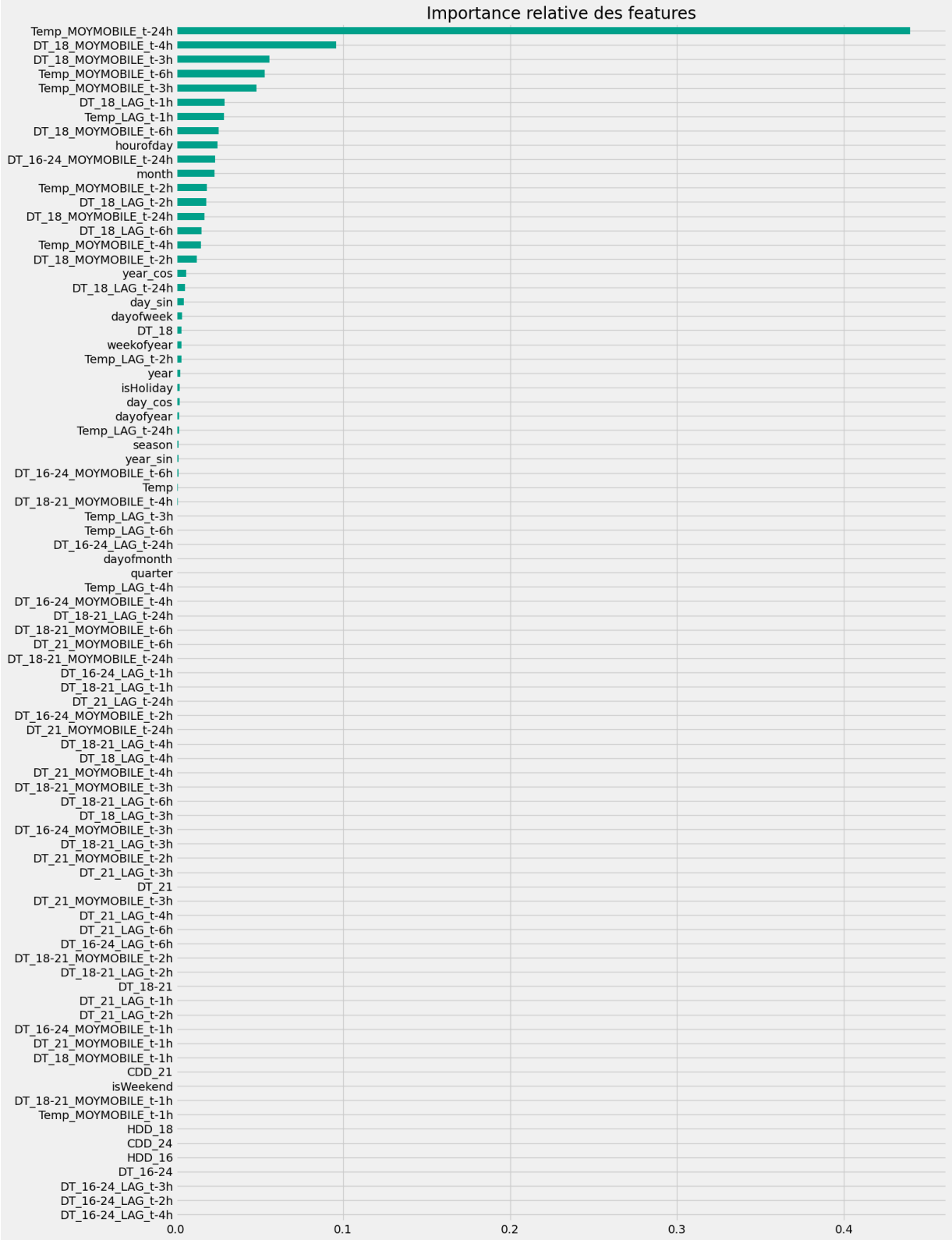
In [27]:

```
rmse_tous_features = reg.best_score
print(
    f"Le meilleur RMSE sur les données d'entraînement est : {rmse_tous_featu
)
```

```
Le meilleur RMSE sur les données d'entraînement est : 1129.95 par rapp
28% mieux.
```

In [28]:

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

fi.sort_values("Importance").plot(
    figsize=(15, 25),
    kind="barh",
    title="Importance relative des features",
    legend=False,
    color=colors_pal[1],
)
plt.show()
```



Nous pouvons bien voir que les moyennes mobiles sont très importantes dans les prédictions, étant 7 fois présents sur les 10 premiers, les lags étant présents 2 fois et l'heure du jour 1 fois seulement.

Gardons seulement les 20 premiers features pour une 3e passe préliminaire et pour voir l'impact sur le résultat.

In [29]:
```
FEATURES_TOP20 = (
    fi.sort_values("Importance", ascending=False)
    .head(20)
    .index.to_list()
)
X_train = train[FEATURES_TOP20]
y_train = train[TARGET]

X_test = test[FEATURES_TOP20]
y_test = test[TARGET]
```

In [30]:
```
n_estim = 2000  # nb trees

reg = xgb.XGBRegressor(
    n_estimators=n_estim,
    early_stopping_rounds=50,
    learning_rate=0.01,
)

reg.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=False,
)
```

Out[30]:

▾ XGBRegressor

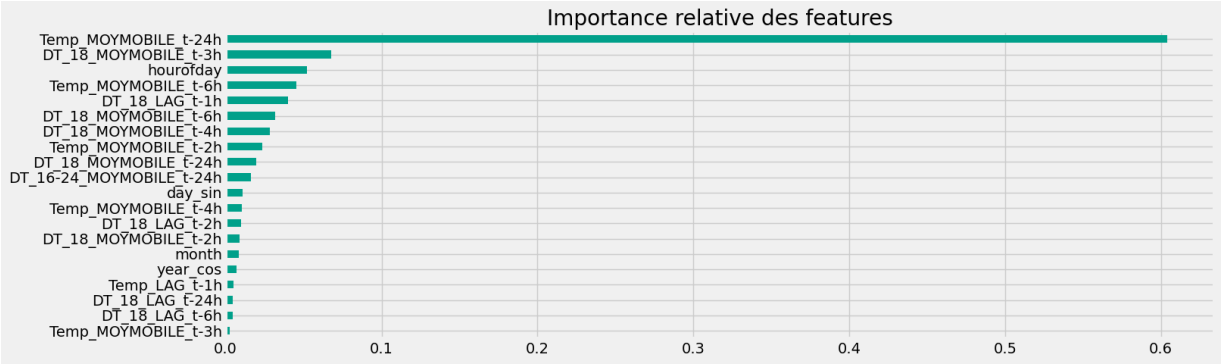
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_ty
pes=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_
type=None,
             interaction_constraints=None, learning_rate=0.01, max_
bin=None,
```

In [31]:
```
rmse_top_20 = reg.best_score
print(
    f"Le meilleur RMSE sur les données d'entraînement est : {rmse_top_20:0.2
)
```

Le meilleur RMSE sur les données d'entraînement est : **1340.58** par rapp
features, soit **-19%** mieux.

In [32]:
```
fi = pd.DataFrame(
    data=reg.feature_importances_,
    index=reg.feature_names_in_,
    columns=["Importance"],
)
fi.sort_values("Importance").plot(
```

```
    figsize=(15, 5),
    kind="barh",
    title="Importance relative des features",
    legend=False,
    color=colors_pal[1],
)
plt.show()
```



L'ordre des features en importance a changé quand on utilise les top 20 précédent. L'erreur a aussi augmenté.

Une chose est certaine, il faut probablement évaluer l'intérêt d'ajouter des features de moyenne mobile avec plus d'heures de décallage.

Deuxième partie - renforcir le modèle

Inspiration

- Vidéo Partie 2 : https://youtu.be/z3ZnOW-S550?si=VcoYQ4S5Zlcqffyr
- Kaggle notebook : https://www.kaggle.com/code/robikscube/pt2-time-series-forecasting-with-xgboost/notebook

Validation croisée (*Time Series Cross Validation*)

Nous désirons tester plusieurs durées différentes d'entraînement afin de maximiser nos données.

Dans une première vague (*fold*), 1 an d'entraînement, 1 an de test.

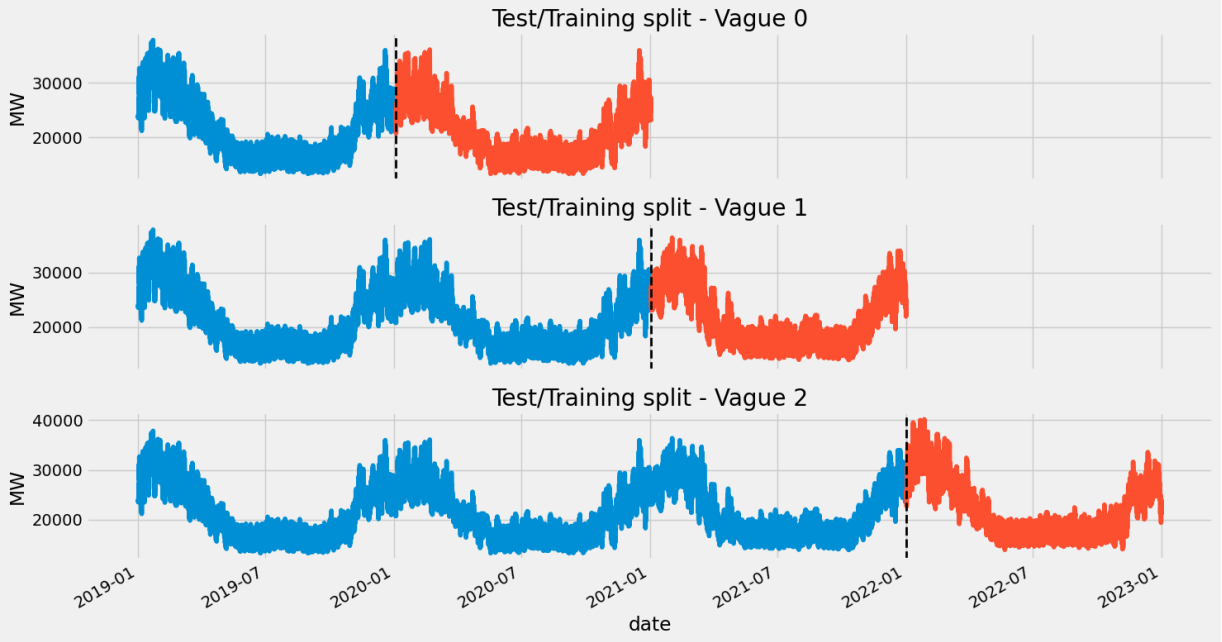
Dans la 2e, 2 ans d'entraînement, 1 an de test. Etc.

test_size = 1 an

In [33]:
```
df = import_and_create_features_no_categorical(
    fin="20221231"
).dropna()
```

In [34]:
```
n_split = 3

tss = TimeSeriesSplit(
    n_splits=n_split, test_size=24 * 365 * 1, gap=24
)
df = df.sort_index()
```

In [35]:
```
fig, axs = plt.subplots(
    n_split, 1, figsize=(15, 8), sharex=True
)

fold = 0
for train_idx, val_idx in tss.split(df):
    train = df.iloc[train_idx]
    test = df.iloc[val_idx]

    train["MW"].plot(
        ax=axs[fold],
        label="Données Entraînement",
        title=f"Test/Training split - Vague {fold}",
        ylabel="MW",
    )
    test["MW"].plot(
        ax=axs[fold],
        label="Données Test",
    )
    axs[fold].axvline(
        test.index.min(),
        color="black",
        ls="--",
        linewidth=2,
    )

    fold += 1
plt.tight_layout()
plt.show()
```



Comme nous n'avons pas énormément des données (seulement 4 ans), nous nous questionnons à savoir si ce sera suffisant. À suivre.

Délai et moyenne mobile sur la cible

- Quelle était la cible (demande) dans le passé.

Cela implique que nous avons les données de la demande en MW en continu (donc impossible de faire des prévisions actuellement, car 2023 n'est pas complète pour l'instant)

Réutilisons les mêmes fonctions que plus tôt, mais avevc la caractéristique 'MW'.

> Dans la démo, il fait un lag de 364 jours (1 an), alors que nous utilisons des lags très court sur la Temp.
> Est-ce possible de faire un lag / moyenne mobile de quelques heures, étant donnné que cela ne fonctionnera pas sur les prévisions (la prévions MW à t + 100h ne connaît pas le MW à t+99h..)
> **À valider**

In [36]:
```
df = create_lag_features(
    df,
    caract=["MW"],
    lags=[364 * 24 * 1, 364 * 24 * 2, 364 * 24 * 3],
)  # 1,2,3 ans
```

Entraînement avec la validation croisée (cross-validation)

In [37]:

```
n_split = 3

tss = TimeSeriesSplit(
    n_splits=n_split, test_size=24 * 365 * 1, gap=24
)
df = df.sort_index()

FEATURES = df.columns.to_list()[1:]  # Enlevons MW
TARGET = "MW"


fold = 0
regressions = []
preds = []
scores = []

for train_idx, val_idx in tss.split(df):
    train = df.iloc[train_idx]
    test = df.iloc[val_idx]

    X_train = train[FEATURES]
    y_train = train[TARGET]

    X_test = test[FEATURES]
    y_test = test[TARGET]

    reg = xgb.XGBRegressor(
        # base_score=0.5,
        # booster="gbtree",
        n_estimators=2000,  # 1000
        early_stopping_rounds=50,
        # objective="reg:linear",
        # max_depth=3,
        learning_rate=0.01,
    )
    reg.fit(
        X_train,
        y_train,
        eval_set=[
            (X_train, y_train),
            (X_test, y_test),
        ],
        verbose=100,
    )

    y_pred = reg.predict(X_test)
    preds.append(y_pred)
    score = np.sqrt(mean_squared_error(y_test, y_pred))
    scores.append(score)
    regressions.append(reg)
```

```
[0]     validation_0-rmse:22375.02775   validation_1-rmse:20963.28497
[100]   validation_0-rmse:8277.24763    validation_1-rmse:7730.30720
[200]   validation_0-rmse:3135.65400    validation_1-rmse:2985.61146
[300]   validation_0-rmse:1292.67473    validation_1-rmse:1394.68190
[400]   validation_0-rmse:683.94432     validation_1-rmse:988.93576
[500]   validation_0-rmse:500.52874     validation_1-rmse:909.43748
[600]   validation_0-rmse:439.64968     validation_1-rmse:888.87286
[700]   validation_0-rmse:410.66039     validation_1-rmse:881.14346
[800]   validation_0-rmse:385.56489     validation_1-rmse:878.86871
[853]   validation_0-rmse:375.98394     validation_1-rmse:879.38664
[0]     validation_0-rmse:21711.00036   validation_1-rmse:21621.26158
[100]   validation_0-rmse:8026.14201    validation_1-rmse:8347.56524
[200]   validation_0-rmse:3046.25831    validation_1-rmse:3548.07401
[300]   validation_0-rmse:1284.19246    validation_1-rmse:1860.01246
[400]   validation_0-rmse:722.47524     validation_1-rmse:1297.19592
[500]   validation_0-rmse:563.43474     validation_1-rmse:1122.85289
[600]   validation_0-rmse:505.12062     validation_1-rmse:1061.12518
[700]   validation_0-rmse:478.62716     validation_1-rmse:1042.22255
[800]   validation_0-rmse:463.01986     validation_1-rmse:1033.85811
[900]   validation_0-rmse:445.85045     validation_1-rmse:1028.90413
[1000]  validation_0-rmse:432.38637     validation_1-rmse:1022.04640
[1100]  validation_0-rmse:419.34583     validation_1-rmse:1014.52224
[1200]  validation_0-rmse:405.51418     validation_1-rmse:1012.82650
[0]     validation_0-rmse:21675.42813   validation_1-rmse:22225.25606
[100]   validation_0-rmse:8009.63427    validation_1-rmse:8740.78358
[200]   validation_0-rmse:3041.65271    validation_1-rmse:3793.58655
[300]   validation_0-rmse:1293.72927    validation_1-rmse:1991.86718
[400]   validation_0-rmse:743.07297     validation_1-rmse:1347.14432
[500]   validation_0-rmse:589.00952     validation_1-rmse:1124.52141
[600]   validation_0-rmse:535.44080     validation_1-rmse:1024.93065
[700]   validation_0-rmse:512.34321     validation_1-rmse:975.74263
[800]   validation_0-rmse:491.72420     validation_1-rmse:965.90043
[900]   validation_0-rmse:475.89876     validation_1-rmse:961.29673
[1000]  validation_0-rmse:463.45158     validation_1-rmse:957.05962
[1100]  validation_0-rmse:452.48765     validation_1-rmse:952.46599
[1200]  validation_0-rmse:439.77777     validation_1-rmse:946.49207
[1300]  validation_0-rmse:428.00242     validation_1-rmse:944.74181
[1400]  validation_0-rmse:417.78456     validation_1-rmse:944.03252
[1500]  validation_0-rmse:407.60428     validation_1-rmse:942.57613
[1600]  validation_0-rmse:398.61652     validation_1-rmse:940.10641
[1700]  validation_0-rmse:388.29095     validation_1-rmse:938.62623
[1800]  validation_0-rmse:379.72643     validation_1-rmse:938.11354
[1838]  validation_0-rmse:376.12260     validation_1-rmse:938.49834
```

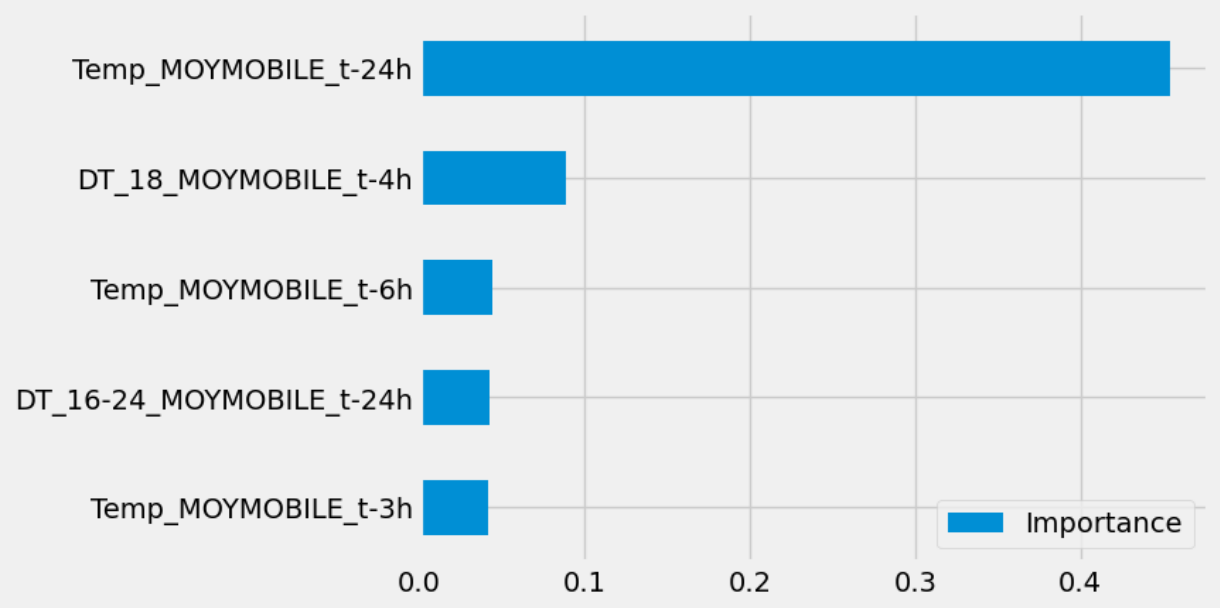
In [38]:

```python
print(f"Résultats moyen {np.mean(scores):0.4f}")
print(f"Résultat de chaque vague :{scores}")

i = 0
for r in regressions:
    i += r.best_iteration
    print("Meilleure itération : ", r.best_iteration)

print(
    "Meilleure itération moyenne", i / len(regressions)
)
```

```
Résultats moyen 942.3950
Résultat de chaque vague :[878.8338201152005, 1010.3996281982209, 937.
Meilleure itération :  803
Meilleure itération :  1150
Meilleure itération :  1789
Meilleure itération moyenne 1247.3333333333333
```

In [39]:
```
fi = pd.DataFrame(
    data=reg.feature_importances_,
    index=reg.feature_names_in_,
    columns=["Importance"],
)
fi.sort_values("Importance", ascending=True).tail(
    5
).plot(kind="barh")
```

Out[39]: <Axes: >



In [40]:
```
fi.loc[fi.index == "MW_LAG_t-8736h"]
```

Out[40]:

	Importance
MW_LAG_t-8736h	0.001048

Le lag d'un an sur la demande semble très peu significatif dans les résultats. Nous pouvons éviter cette caractéristique, qui pourrait nous empêcher de faire des prévisions à plus long terme.

Prédictions dans le futur

Afin de profiter de l'ensemble des données que nous avons, nous entraînons le modèle sur les 4 années disponibles avec les mêmes paramètres que précedemment.

Entraînement sur l'ensemble des données disponibles

In [41]:
```
(
    df,
    InfoDates,
) = import_and_create_features_no_categorical(
    lags=[
        1,
        2,
        3,
        4,
        6,
        24,
        364 * 24 * 1,
        364 * 24 * 2,
        364 * 24 * 3,
    ],
    fenetres=[1, 2, 3, 4, 6, 8, 12, 16, 24],
    fin="20221231",
    getInfoDate=True,
)

df = df.dropna()

print(InfoDates)
```

```
{
   'dateMin': Timestamp('2018-01-01 00:00:00'),
   'dateMax': Timestamp('2023-12-18 23:00:00'),
   'dateMaxMW': Timestamp('2023-12-03 00:00:00')
}
```

In [42]:
```
FEATURES = df.columns.to_list()[
    1:
]  # Enlevons  en première colonne
TARGET = "MW"

X_all = df[FEATURES]
y_all = df[TARGET]

reg = xgb.XGBRegressor(
    # base_score=0.5,
    # booster='gbtree',
    n_estimators=2000,
    # objective='reg:linear',
    # max_depth=3,
    learning_rate=0.01,
)
reg.fit(
    X_all,
    y_all,
    eval_set=[(X_all, y_all)],
```

```
    verbose=250,
)
```

```
[0]     validation_0-rmse:22284.74989
[250]   validation_0-rmse:1978.80480
[500]   validation_0-rmse:546.76785
[750]   validation_0-rmse:436.06390
[1000]  validation_0-rmse:389.19289
[1250]  validation_0-rmse:356.68995
[1500]  validation_0-rmse:329.13842
[1750]  validation_0-rmse:307.49308
[1999]  validation_0-rmse:289.17604
```

Out[42]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_ty
pes=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_
type=None,
             interaction_constraints=None, learning_rate=0.01, max_
bin=None,
```

DF avec toutes les dates disponibles pour prédictions

On va chercher toutes les dates futures, ou dans notre cas, à partir du **12 novembre 2023**, où nous commençons à avoir de l'historique de la demande en MW + période d'environ 16 jours dans le futur où nous n'en avons pas.

In [43]:

```
# DF avec données historiques + 16 jours de prédictions de températures
df_all_time = (
    import_and_create_features_no_categorical(
        lags=[
            1,
            2,
            3,
            4,
            6,
            24,
            364 * 24 * 1,
            364 * 24 * 2,
            364 * 24 * 3,
        ],
        fenetres=[1, 2, 3, 4, 6, 8, 12, 16, 24],
    )
)

df_all_time["d"] = df_all_time.index
df_all_time["isFuture"] = df_all_time["d"].apply(
    lambda x: x > InfoDates.get("dateMaxMW")
```

```
)

## Pour avoir de la demande à comparer avec prédictions
df_all_time["isFuture_but_MW_History"] = df_all_time[
    "d"
].apply(lambda x: x > datetime(2023, 11, 12))

df_all_time = df_all_time.drop(columns=["d"])

df_all_time = df_all_time.loc[
    df_all_time.Temp.notna()
]  # Enlève données Temp manquantes à la fin du df

df_all_time.tail(5)
```

Out[43]:

date	MW	Temp	hourofday	quarter	year	dayofyear	dayofmonth	weekofyear
2023-12-18 15:00:00	NaN	-0.6	15	4	2023	352	18	51
2023-12-18 16:00:00	NaN	-0.4	16	4	2023	352	18	51
2023-12-18 17:00:00	NaN	-0.4	17	4	2023	352	18	51
2023-12-18 18:00:00	NaN	-0.4	18	4	2023	352	18	51
2023-12-18 19:00:00	NaN	-0.5	19	4	2023	352	18	51

In [44]:

```
# Gardons les dates futures (ou dans ce cas, les dates d'historique de demar
future_w_features = df_all_time.query(
    "isFuture_but_MW_History"
).copy()

# Réalisons les prédictions sur les données "futures"
future_w_features["pred"] = reg.predict(
    future_w_features[FEATURES]
)

future_w_features[
    [
        "MW",
        "isFuture",
        "isFuture_but_MW_History",
        "pred",
```

```
    ]
]
```

Out[44]:

date	MW	isFuture	isFuture_but_MW_History	pred
2023-11-12 01:00:00	21648.75	False	True	21942.791016
2023-11-12 02:00:00	21683.00	False	True	21959.664062
2023-11-12 03:00:00	21834.50	False	True	22051.394531
2023-11-12 04:00:00	22058.50	False	True	22054.923828
2023-11-12 05:00:00	22467.50	False	True	22416.337891
...	...	...	...	...
2023-12-18 15:00:00	NaN	True	True	26606.419922
2023-12-18 16:00:00	NaN	True	True	27213.646484
2023-12-18 17:00:00	NaN	True	True	27525.701172
2023-12-18 18:00:00	NaN	True	True	27511.392578
2023-12-18 19:00:00	NaN	True	True	27144.457031

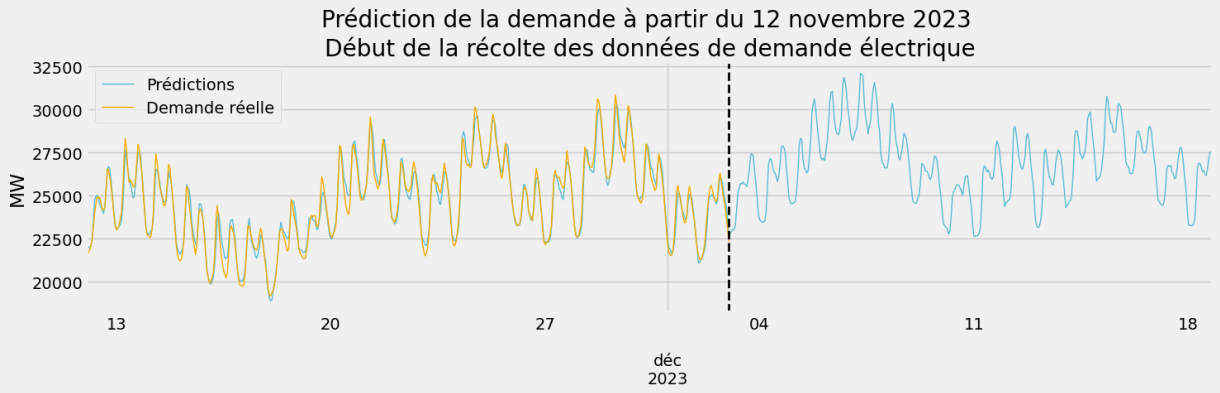
883 rows × 4 columns

Visualisation des résultats

In [50]:

```
ax = future_w_features["pred"].plot(
    figsize=(15, 5),
    color=colors_pal[4],
    ms=1,
    lw=1,
    title="Prédiction de la demande à partir du 12 novembre 2023 \nDébut de
    label="Prédictions",
    xlabel="",
    ylabel="MW",
)

future_w_features["MW"].plot(
    ax=ax,
    color=colors_pal[2],
    ms=1,
    lw=1,
    label="Demande réelle",
    xlabel="",
)

ax.axvline(
    datetime(2023, 12, 3),
    color="black",
    ls="--",
```

```
    linewidth=2,
)

ax.legend()
plt.tight_layout()
plt.show()
```



Les prédictions sont débutées à partir du 12 nov.

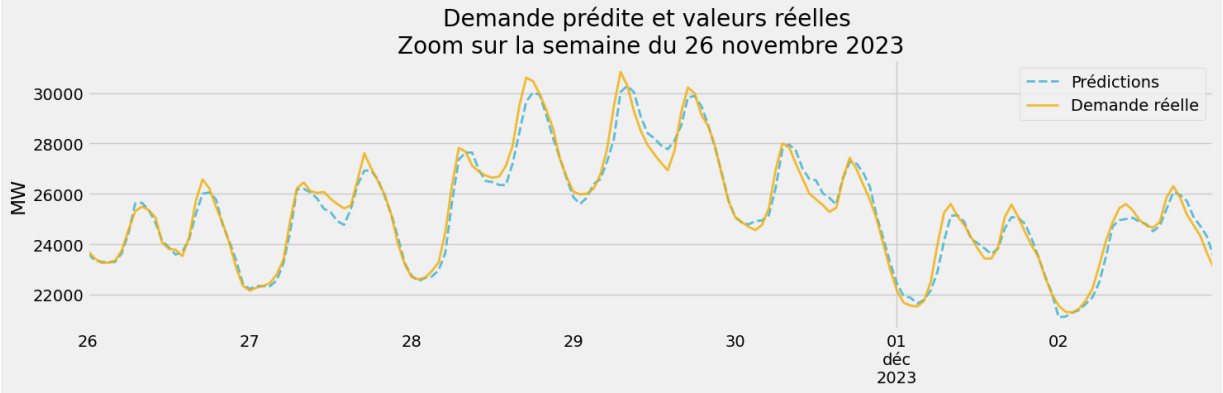
Nous avons un historique à partir du 12 jusqu'au 3 décembre, que nous pouvons comparer avec les données réelles enregistrées.

Au-delà du 3 décembre, nous avons seulement des prévisions en fonction des prévisions météo.

Effectuons une visualisation sur quelques jours seulement pour apprécier le résultat.

In [46]:
```
future_w_features_zoom = future_w_features[
    "20231126":"20231202"
].copy()

ax = future_w_features_zoom["pred"].plot(
    figsize=(15, 5),
    color=colors_pal[4],
    ms=1,
    lw=2,
    ls="--",
    title="Demande prédite et valeurs réelles \nZoom sur la semaine du 26 no
    label="Prédictions",
    xlabel="",
    ylabel="MW",
)

future_w_features_zoom["MW"].plot(
    ax=ax,
    color=colors_pal[2],
    ms=1,
    lw=2,
    alpha=0.8,
    label="Demande réelle",
    xlabel="",
)
```

```
ax.axvline(
    datetime(2023, 12, 3),
    color="black",
    ls="--",
    linewidth=2,
)

ax.legend()
plt.tight_layout()
plt.show()
```



Nous voyons que les valeurs prédites et réelles sont très près une de l'autre.

Calcul des erreurs

In [48]:
```
_ = calcul_erreurs(
    future_w_features,
    nomColPrediction="pred",
    nomColReel="MW",
)
```

```
Le MSE est de 174049.8, le RMSE est de 417.2 et le MAE de 313.3 pour u
```

Nous pouvons voir que les prévisions sont très bonnes en comparaison avec la demande réelle enregistrée. Pour une simple régression, le RMSE était à près de 2400. C'est aussi très visible sur le graphique que la prédiction suit très bien la valeur réelle de la demande.

Apprentissages

Nous avons de bons résultats préliminaires avec ce modèle, pour être en mesure de faire des prévisions à court terme.

Les délais (*lags*) sur la demande en MW en tant que tel n'est pas significative en termes d'importance dans le modèle. Nous décidons de ne pas l'utiliser dans le futur, car elle pourrait potentiellement nous empêcher de faire des prévisions à plus long terme.

Si nous décidons de poursuivre avec XGBoost, nous pourrons réaliser les tâches suivantes.

À faire

- [ ] Cross-validation : à explorer un peu plus si requis
- [ ] Test plusieurs features
- [ ] Création de nouveaux features en fonction de ceux qui seront les meilleurs
- [ ] Tests paramètres XGBoost
  - https://365datascience.com/tutorials/python-tutorials/xgboost-lgbm/#:~:text=Step%208%3A%20Tune%20the%20XGBoost%20Model
  - https://optuna.org/ : Optimisation automatisée des paramètres