C_c_1 Optimisation du modèle

Après avoir choisi le modèle de **XGBoost** pour réaliser la fin du travail et la visualisation, il faut maintenant trouver les hyperparamètres du modèle qui nous permettra les meilleurs résultats.

Pour ce faire, nous utiliserons le cadre d'automatisation de l'optimisation des hyperparemètres **Optuna**

In [15]:
```
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 optuna

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")

print("Version XGBoost :", xgb.__version__)
```

```
Version XGBoost : 1.7.6
```

In [18]:
```
# %load_ext jupyter_black

import black
```

```
import jupyter_black

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

Création du modèle de base

Nous recréons la séquence pour définir notre modèle de base.

Import et création de la liste des caractéristiques

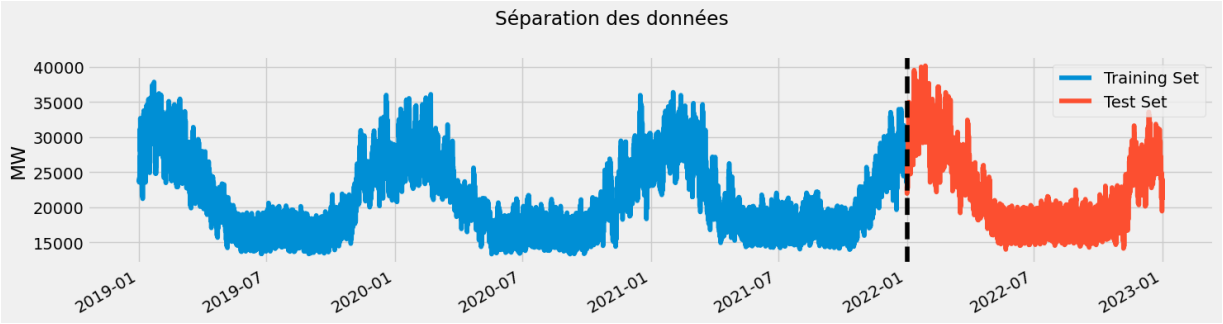
In [19]:
```
(
    df,
    InfoDates,
) = import_and_create_features_no_categorical(
    fenetres=[1, 2, 3, 4, 6, 8, 12, 16, 24],
    fin="20221231",
    getInfoDate=True,
)

df = df.dropna()

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

print(
    f"Nous avons {len(FEATURES)} caractéristiques dans le modèle après la cr
)
print(InfoDates)
```

```
Nous avons 99 caractéristiques dans le modèle après la création de cel
{
    'dateMin': Timestamp('2018-01-01 00:00:00'),
    'dateMax': Timestamp('2023-12-24 23:00:00'),
    'dateMaxMW': Timestamp('2023-12-09 06:00:00')
}
```

Séparer les données en un ensemble d'entraînement et de tests

In [20]:
```
date_slit = "2022-01-01"

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

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

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

# Visualisation
fig, ax = plt.subplots(figsize=(15, 4))
train["MW"].plot(ax=ax, label="Entraînement", ylabel="MW", xlabel="")
test["MW"].plot(ax=ax, label="Test", xlabel="")
ax.legend(["Training Set", "Test Set"])
ax.axvline(date_slit, color="black", ls="--")
fig.suptitle("Séparation des données")
fig.tight_layout()
plt.show()
```



Graphique des prédictions

In [21]:
```
def graphiques_pred(test):
    fig, ax = plt.subplots(figsize=(15, 5))
    test["MW"].plot(
        ax=ax,
        label="Valeurs réelles",
        ylabel="MW",
        xlabel="",
    )
    test["prediction"].plot(
        ax=ax,
        label="Prédictions",
        xlabel="",
    )
    ax.legend()
    fig.suptitle("Prédictions et données réelles - 2022")
    plt.show()

    test_fev = test["20220210":"20220220"]

    fig, ax = plt.subplots(figsize=(15, 5))
    test_fev["MW"].plot(
        ax=ax,
        label="Valeurs réelles",
        ylabel="MW",
        xlabel="",
        alpha=0.7,
    )
    test_fev["prediction"].plot(
```

```
        ax=ax,
        label="Prédictions",
        xlabel="",
        style="--",
        alpha=0.8,
    )
    ax.legend()
    fig.suptitle("Prédictions et données réelles - Février 2022")
    plt.show()
```

Création du modèle XGBoost, prédictions et calculs des erreurs

Créons une cellule qui aura tous les paramètres jusqu'au calcul des métriques d'évaluation, de manière à avoir un test facile (possibilité de changer les paramètres et les tester)

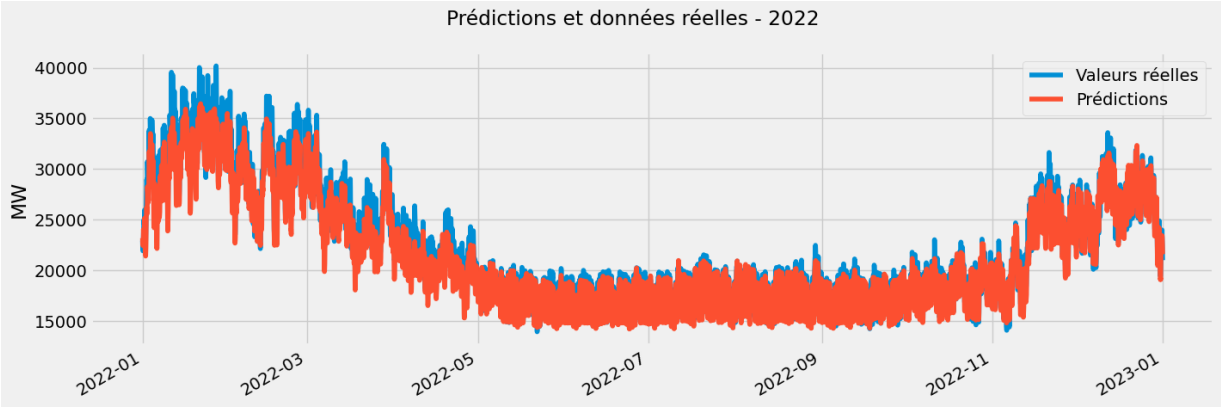
Références hyperparamètres

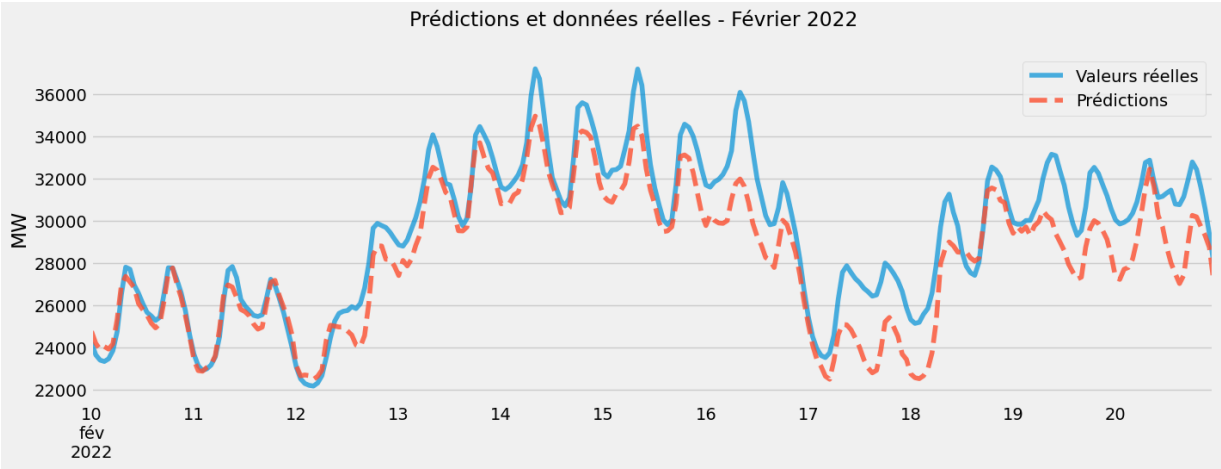
https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning

In [22]:
```
reg = xgb.XGBRegressor(
    base_score=0.5,
    booster="gbtree",  # 3 options  : gbtree, gblinear or dart.
    n_estimators=2000,
    objective="reg:squarederror",
    max_depth=4,  # defaut 6, typique 3-10,
    learning_rate=0.01,  # Typical final values : 0.01-0.2, alias eta
    gamma=0,  # defaut 0, range 0-inf, alias min_split_loss
    min_child_weight=1,  # defaut 1, range 0-inf
    max_delta_step=0,  # defaut 0, range 0-inf, typ 1-10
    subsample=1,  # defaut 1, typ 0.5-1, range 0-1
    reg_lambda=1,  # defaut 1, , typ 1+
    alpha=0,  # def 0, typ 0+
    tree_method="auto",  # def auto, Choices: auto, exact, approx, hist, gpu
    predictor="cpu_predictor",  #      Force XGBoost to use specific predict
)

reg.fit(
    X_train,
    y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=250,
)

test["prediction"] = reg.predict(X_test)

erreurs = calcul_erreurs(test, nomColPrediction="prediction", nomColReel="MW

graphiques_pred(test)
```

```
[0]     validation_0-rmse:21673.92558   validation_1-rmse:22647.72443
[250]   validation_0-rmse:2029.34470    validation_1-rmse:2868.49570
[500]   validation_0-rmse:793.20803     validation_1-rmse:1330.78373
[750]   validation_0-rmse:698.27584     validation_1-rmse:1185.70800
[1000]  validation_0-rmse:654.79611     validation_1-rmse:1155.99193
[1250]  validation_0-rmse:623.05336     validation_1-rmse:1141.90156
[1500]  validation_0-rmse:598.46519     validation_1-rmse:1133.21299
[1750]  validation_0-rmse:577.26883     validation_1-rmse:1127.33448
[1999]  validation_0-rmse:559.28083     validation_1-rmse:1124.60957
```

Le MSE est de **1264746.7**, le RMSE est de **1124.6** et le MAE de **790.5** pour





Utilisation de Optuna

Le cadre **Optuna** permet de tester plusieurs hyperparamètres dans des plages de données précises afin de trouver les meilleurs combinaisons.

In [22]:
```python
def objective(trial):
    base_score = trial.suggest_float("base_score", 0.3, 0.7)
    n_estimators = trial.suggest_int("n_estimators", 1000, 3000)
    max_depth = trial.suggest_int("max_depth", 3, 6)  # defaut 6, typique 3-
    learning_rate = trial.suggest_float(
        "learning_rate",
        0.005,
        0.2,
        log=True,
    )  # Typical final values : 0.01-0.2, alias eta
    gamma = trial.suggest_float(
```

```
        "gamma", 0, 5
    )  # defaut 0, range 0-inf, alias min_split_loss
    min_child_weight = trial.suggest_int(
        "min_child_weight", 3, 7
    )  # defaut 1, range 0-inf
    max_delta_step = trial.suggest_int(
        "max_delta_step", 0, 2
    )  # defaut 0, range 0-inf, typ 1-10
    subsample = trial.suggest_float(
        "subsample", 0.5, 1
    )  # defaut 1, typ 0.5-1, range 0-1
    reg_lambda = trial.suggest_float("reg_lambda", 6, 8)  # defaut 1, , typ
    alpha = trial.suggest_float("alpha", 6, 10)  # def 0, typ 0+

    booster = "gbtree"

    reg = xgb.XGBRegressor(
        objective="reg:squarederror",
        booster=booster,
        tree_method="auto",
        predictor="cpu_predictor",
        early_stopping_rounds=50,
        base_score=base_score,
        n_estimators=n_estimators,
        max_depth=max_depth,
        learning_rate=learning_rate,
        gamma=gamma,
        min_child_weight=min_child_weight,
        max_delta_step=max_delta_step,
        subsample=subsample,
        reg_lambda=reg_lambda,
        alpha=alpha,
    )

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

    return reg.best_score
```

Création d'une étude

In [ ]:
```
study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)
```

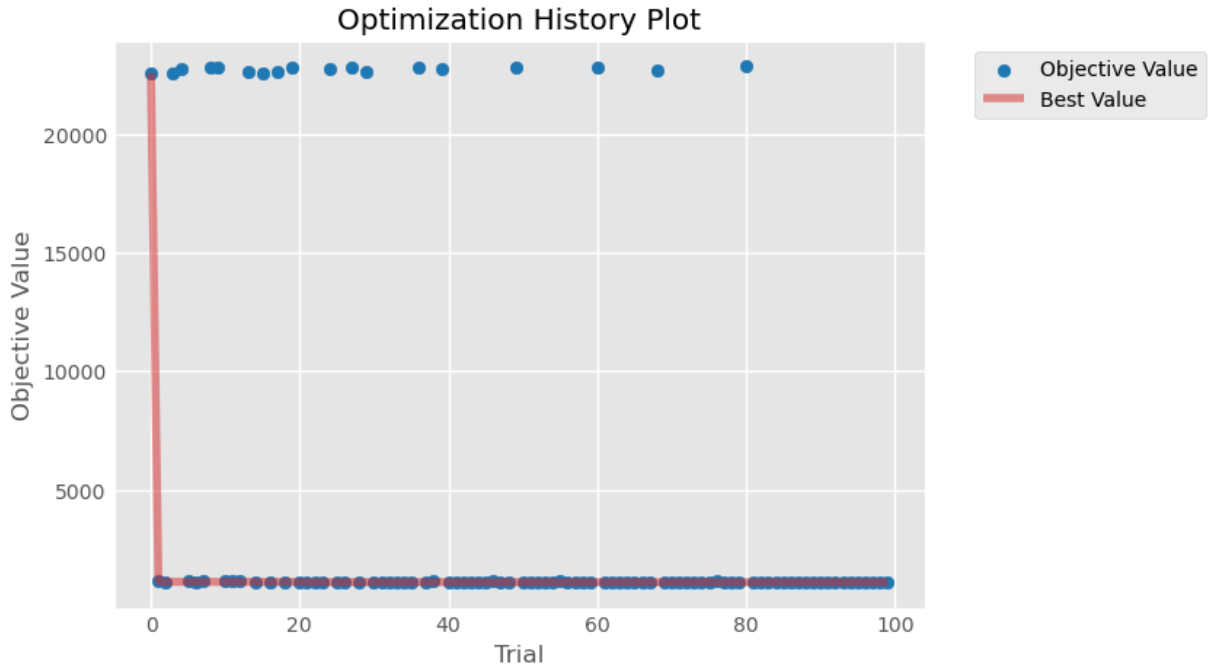
In [25]:
```
best_params = study.best_params
best_value = study.best_value

print(
    f"La meilleur valeur obtenue est {best_value:0.2f} avec les paramètres s
```

```
)
print(best_params)
```

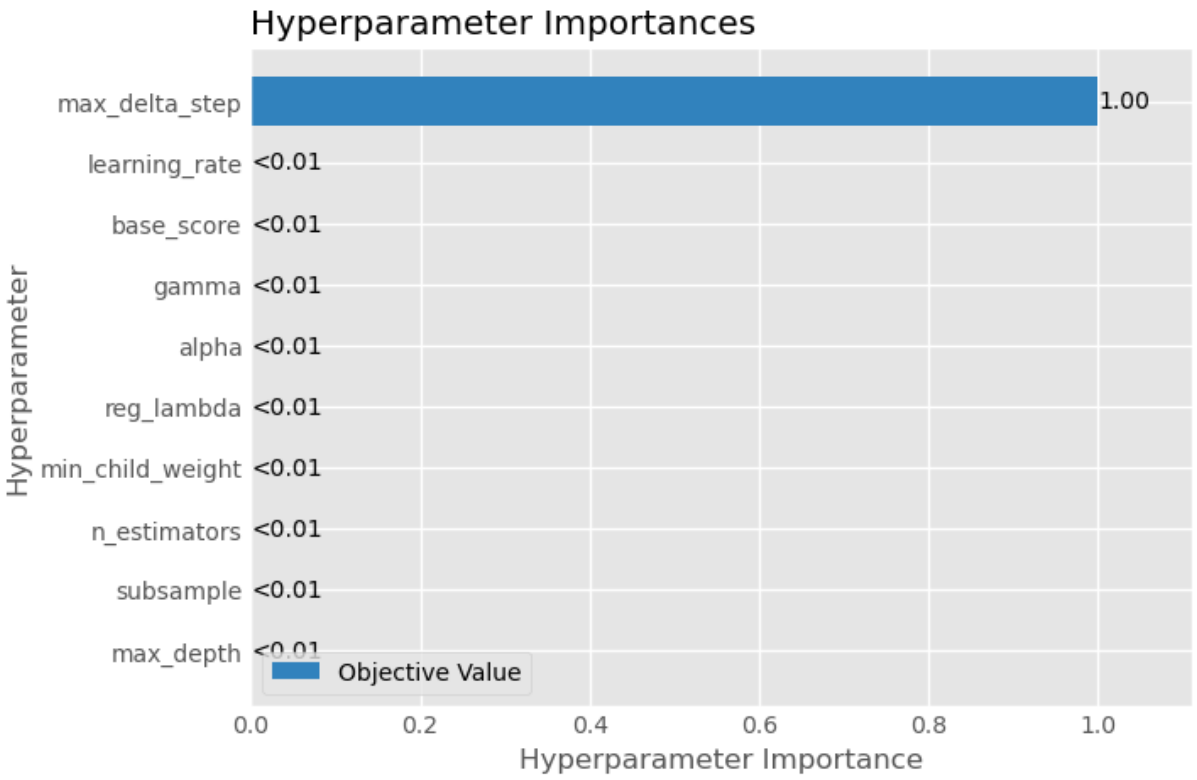
```
La meilleur valeur obtenue est 1101.59 avec les paramètres suivants :
{
    'base_score': 0.48744089966338155,
    'n_estimators': 2947,
    'max_depth': 4,
    'learning_rate': 0.03850106464653303,
    'gamma': 0.531477152369038,
    'min_child_weight': 5,
    'max_delta_step': 0,
    'subsample': 0.6022410508743746,
    'reg_lambda': 7.911950992056376,
    'alpha': 8.070212966705563
}
```

Visualisation du processus

In [26]:
```
optuna.visualization.matplotlib.plot_optimization_history(study)
plt.show()
```



In [27]:
```
optuna.visualization.matplotlib.plot_param_importances(study)
plt.show()
```



Enregistrer le modèle

In [34]:
```
reg_best.save_model("../models/C_c_1_model.json")
```

Essais

Nous avons réalisé de nombreux essais en ajustant les plages de paramètres et le nombre d'essais. Voici quelques-uns des résultats :

Nous enregistrons le meilleur modèle : `../models/optuna_best_model.json`

2023-12-08

La meilleur valeur obtenue est 1101.59 avec les paramètres suivants :

```
{
    'base_score': 0.48744089966338155,
    'n_estimators': 2947,
    'max_depth': 4,
    'learning_rate': 0.03850106464653303,
    'gamma': 0.531477152369038,
    'min_child_weight': 5,
    'max_delta_step': 0,
    'subsample': 0.6022410508743746,
    'reg_lambda': 7.911950992056376,
```

```
    'alpha': 8.070212966705563
}
```

Avec 500 essais :

RMSE = 1096

```
{
    'base_score': 0.6058307333735515,
    'n_estimators': 2766,
    'max_depth': 5,
    'learning_rate': 0.08720899271197992,
    'gamma': 0.21612873986486347,
    'min_child_weight': 7, 'max_delta_step': 0,
    'subsample': 0.8784289534737497,
    'reg_lambda': 6.084923990944192,
    'alpha': 7.371073701933941
}
```

Avec 250 essais :

```
La meilleure valeur obtenue est 1106.59 avec les paramètres
suivants :
{
    'base_score': 0.5383123166214515,
    'n_estimators': 1448,
    'max_depth': 6,
    'learning_rate': 0.011778536074849822,
    'gamma': 0.9752416527353414,
    'min_child_weight': 7,
    'max_delta_step': 0,
    'subsample': 0.7893392775122215,
    'reg_lambda': 6.542521027518733,
    'alpha': 7.947019704214357
}
```

Avec 500 essais :

```
La meilleur valeur obtenue est 1092.93 avec les paramètres
suivants :
{
    'base_score': 0.48734833354672796,
    'n_estimators': 2998,
    'max_depth': 5,
    'learning_rate': 0.14107499643561294,
    'gamma': 1.736940443998144,
    'min_child_weight': 7,
    'max_delta_step': 0,
    'subsample': 0.5586013331650149,
    'reg_lambda': 6.084804186248017,
```

```
    'alpha': 8.837999369503603
}
```

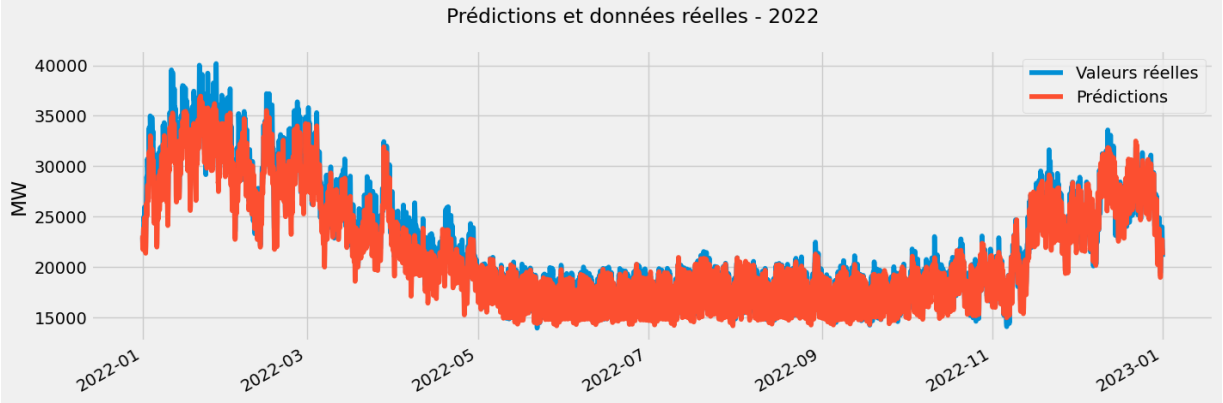
Prédictions avec les meilleurs paramètres - Modèle sauvegardé

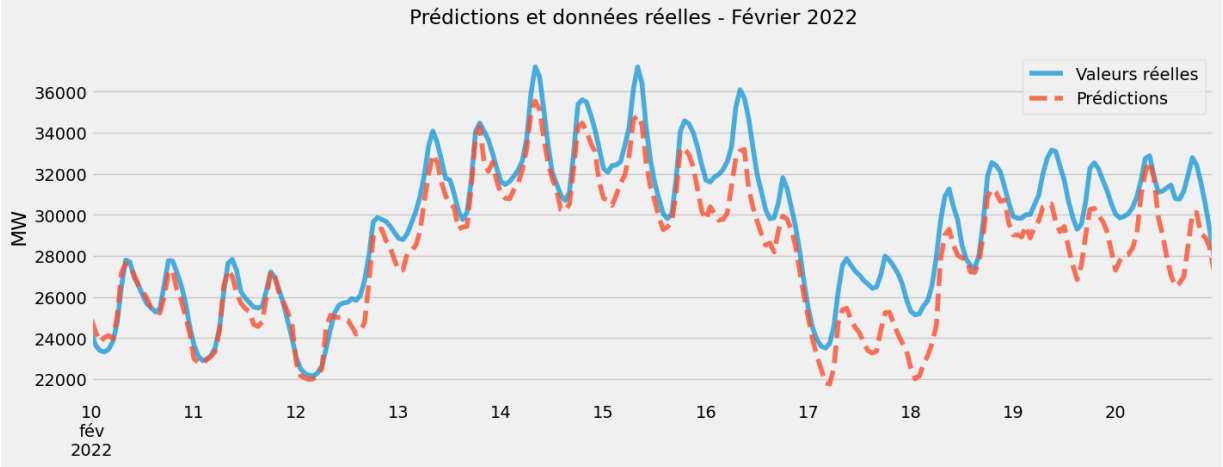
Nous chargeons le meilleur modèle et effectuons des prédictions sur 2022.

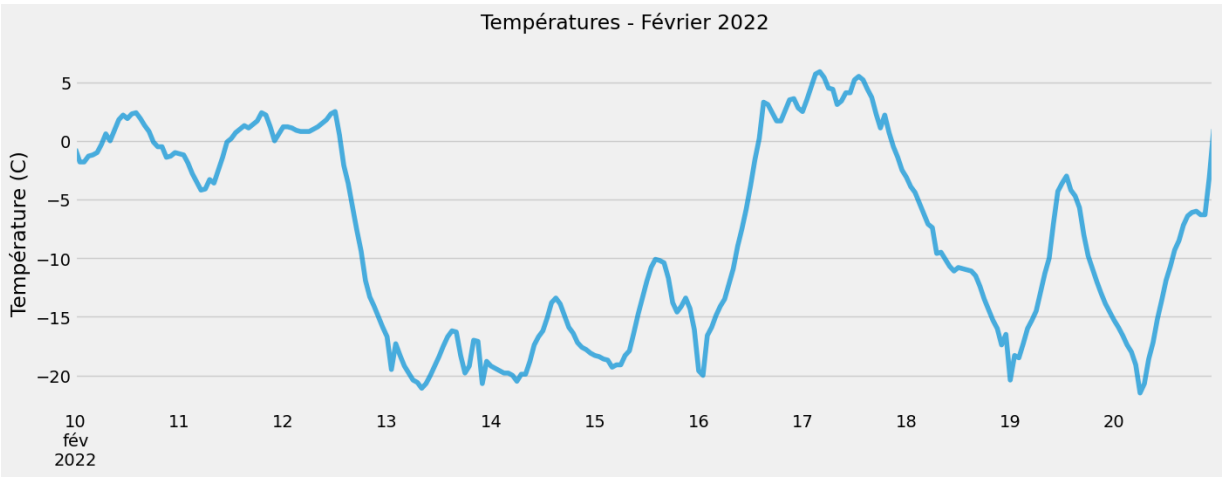
In [13]:
```
reg_best = xgb.XGBRegressor()
reg_best.load_model("../models/optuna_best_model.json")

print(f"Meilleure itération : {reg_best.best_iteration}")

test["prediction"] = reg_best.predict(
    X_test, iteration_range=(0, reg_best.best_iteration + 1)
)

erreurs = calcul_erreurs(test, nomColPrediction="prediction", nomColReel="MW

graphiques_pred(test)

# Températures
test_fev = test["20220210":"20220220"]

fig, ax = plt.subplots(figsize=(15, 5))
test_fev["Temp"].plot(
    ax=ax,
    ylabel="Température (C)",
    xlabel="",
    alpha=0.7,
)

fig.suptitle("Températures - Février 2022")
plt.show()
```

```
Meilleure itération : 156
```

Le MSE est de **1194505.5**, le RMSE est de **1092.9** et le MAE de **776.0** pour





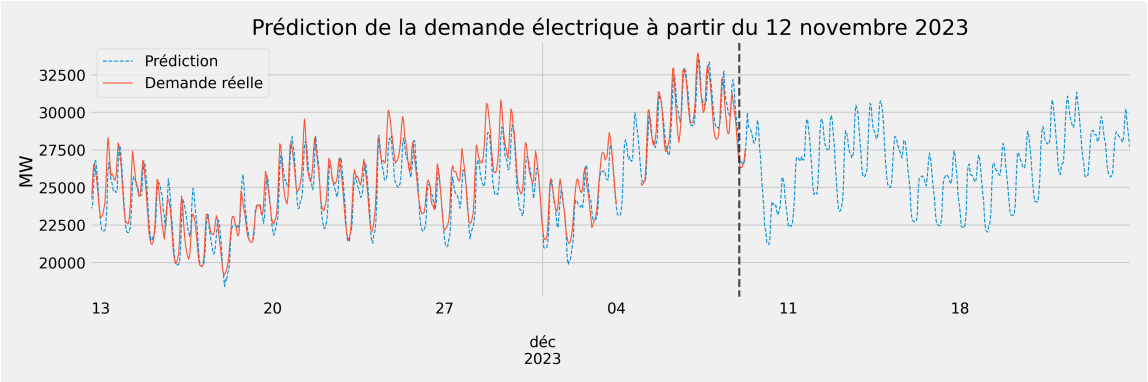


Nous pouvons voir l'amélioration avec un RMSE à 1092 versus des valeurs de 1102 à 1124 initialement. Par contre, il a fallu au moins une heure de calculs pour y arriver. Nous n'arrivons pas à diminuer l'erreur significativement en augmentant le nombre d'essais.

Au point de vue graphique, nous pouvons voir que les prédictions ont toujours un peu de difficulté à suivre les demandes extrêmes, comme pour le matin du 14 février. Il semble avoir un peu de difficulté à suivre les grandes variations de température, par exemple, l'augmentation de 25 degrés C du 16 au 17 février.

Prochaines étapes

Nous préparons les prédictions dans le fichier `src/models/predict_model.py` en le transformant en fonction, qui sera assez modulable pour être réutilisée dans notre outil de visualisation, ce qui nous permet de réaliser ce graphique de prédiction :


Prédiction de la demande électrique à partir du 12 novembre 2023
Prédiction
Demande réelle
MW
32500
30000
27500
25000
22500
20000
13
20
27
04
11
18
déc
2023