Prévoir la consommation électrique pour favoriser une distribution énergétique optimale.

Eloi Kling - Téodore Autuly

Problématiques

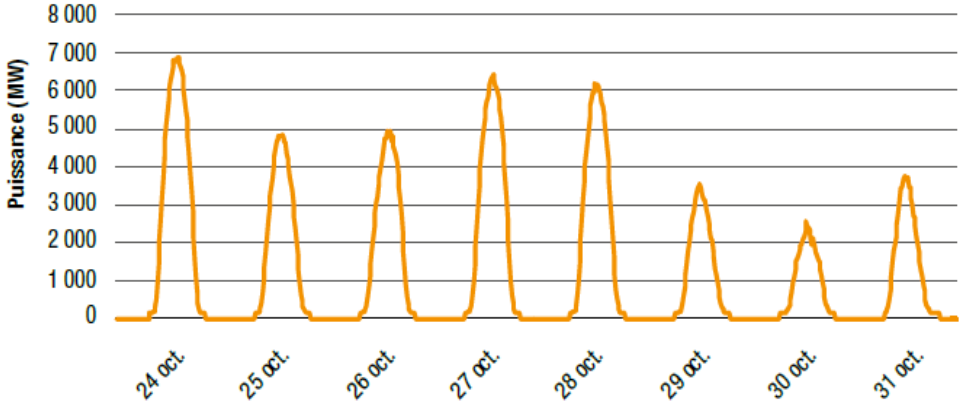
- Comment prévoir efficacement la consommation électrique à venir ?
- Quels modèles de Machine Learning sont judicieux pour cette étude ?
- Comment fonctionnent-ils ?
- Quels sont les paramètres prédominants affectant les données ?

Contexte

- Besoin d'optimiser la production et la distribution d'électricité.
- Toute électricité produite en excès est soit stockée, soit perdue.
- Stockage de l'énergie inefficace et source de nombreuses pertes.
- Il faut produire en fonction de la demande.

Contexte

Figure 1.b – Production solaire horaire en France métropolitaine la semaine du 24 au 31 octobre 2021



Source: *La fabrique de l'industrie: couvrir nos besoins énergétiques*

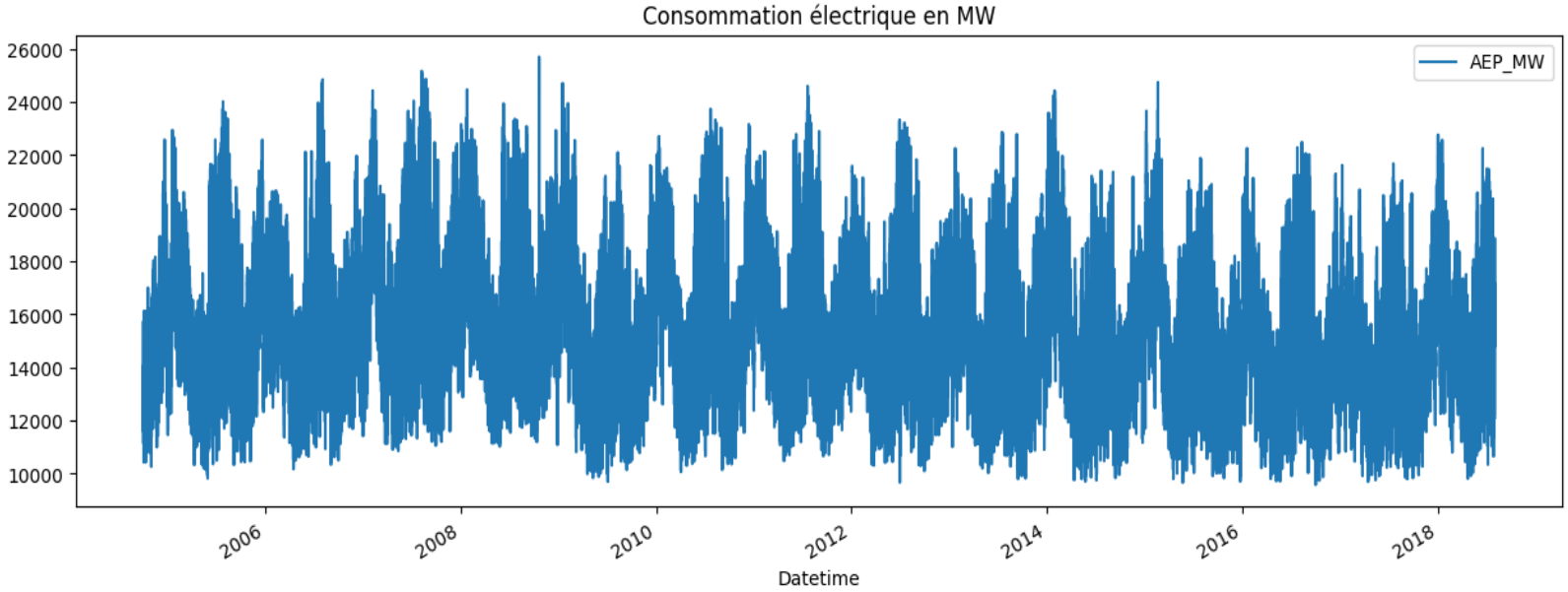
Base de données

Consommation électrique aux Etats-Unis

- **12 ans de données :** 31 décembre 2004 au 2 janvier 2018
- **Fréquence:** heure par heure
- **2 colonnes**
- **121 273 lignes**

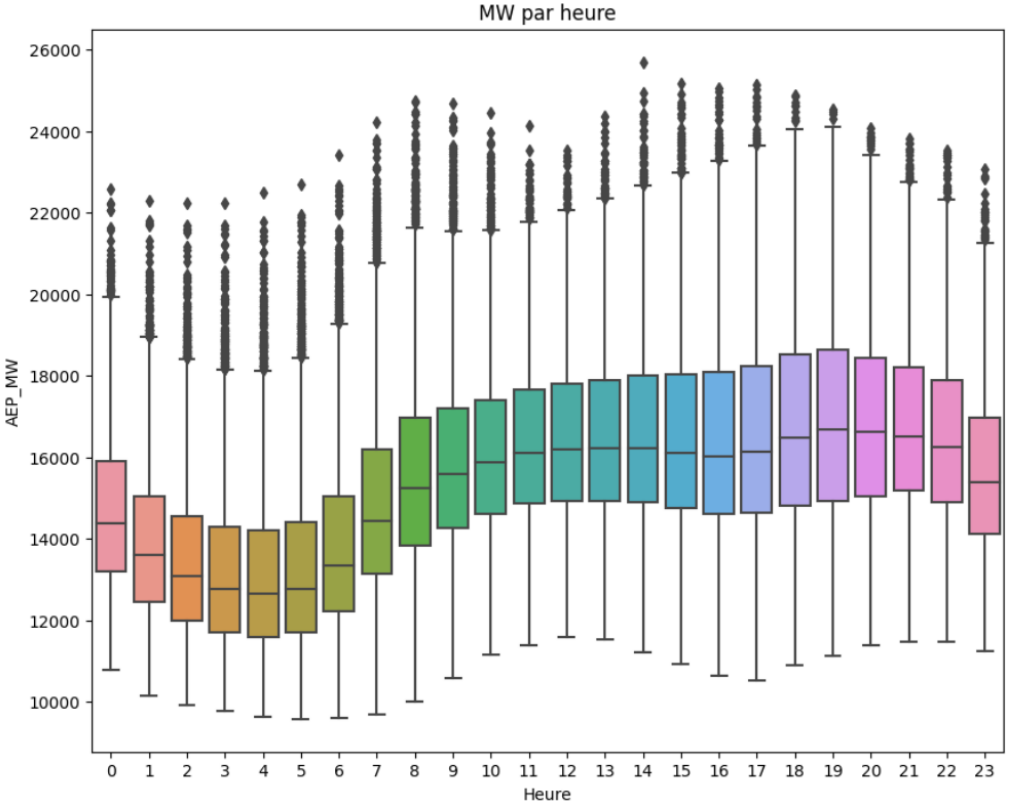
Base de données

Affichage des données



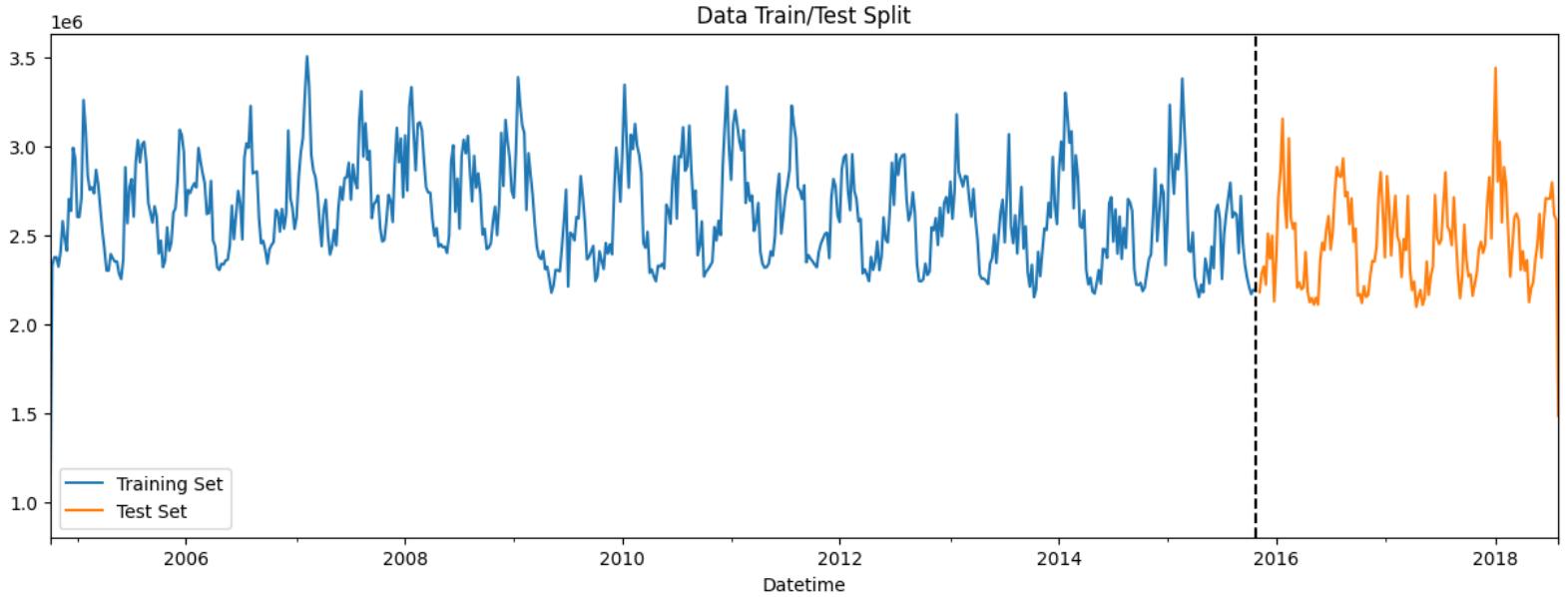
Base de données

Affichage des données

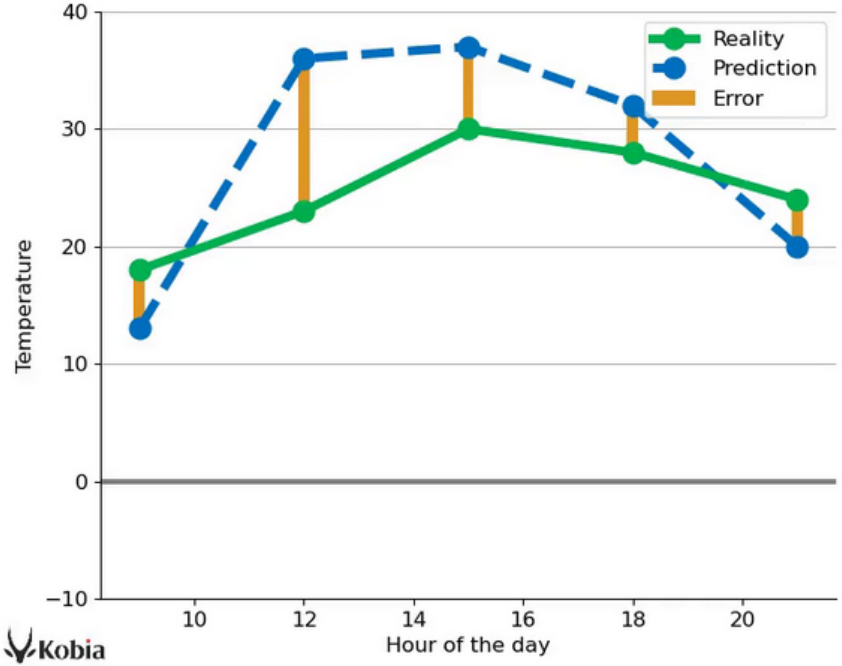


Partage des données : Train / Test

- 80% ➟ Entraînement du modèle
- 20% ➟ Test du modèle



Erreur / évaluation des modèles



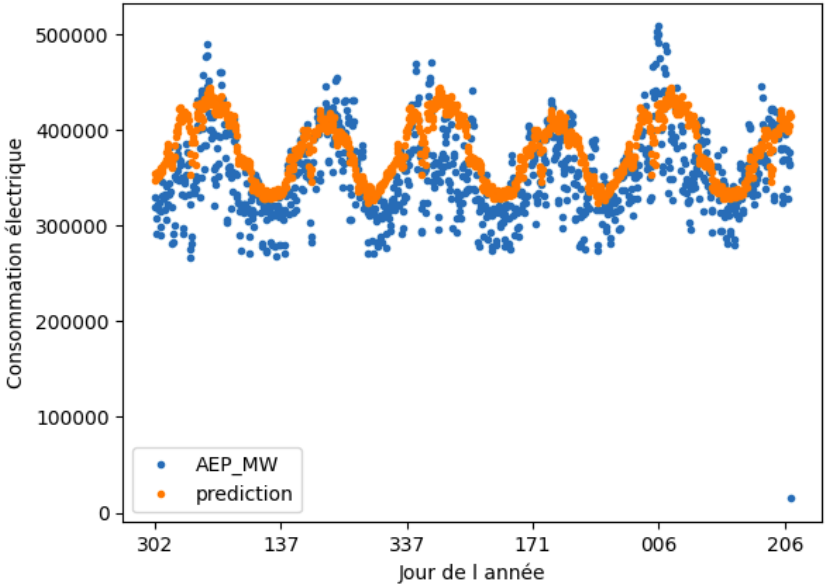
https://kobia.fr/regression-metrics-quelle-metrique-choisir/

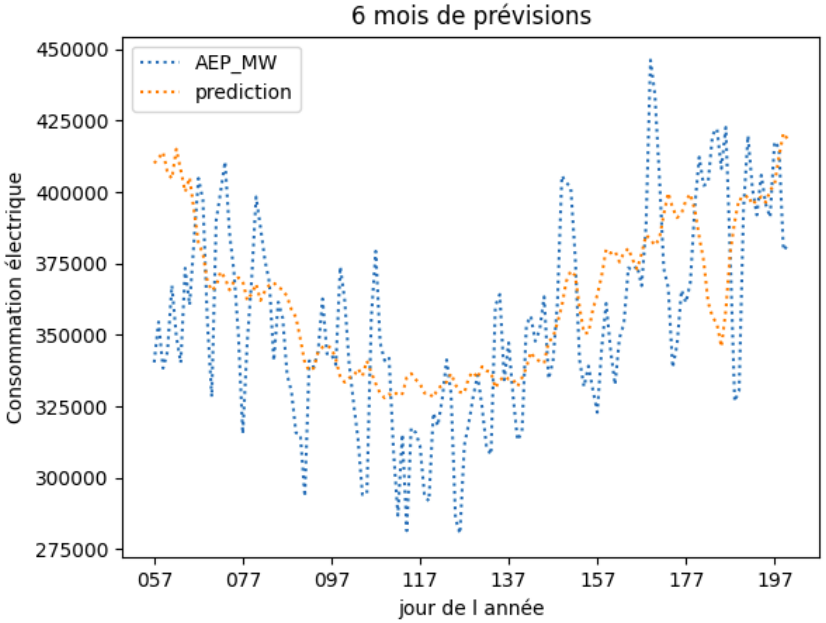
RMSE : Root Mean Squared Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Modèle statistique

Consommation par jour





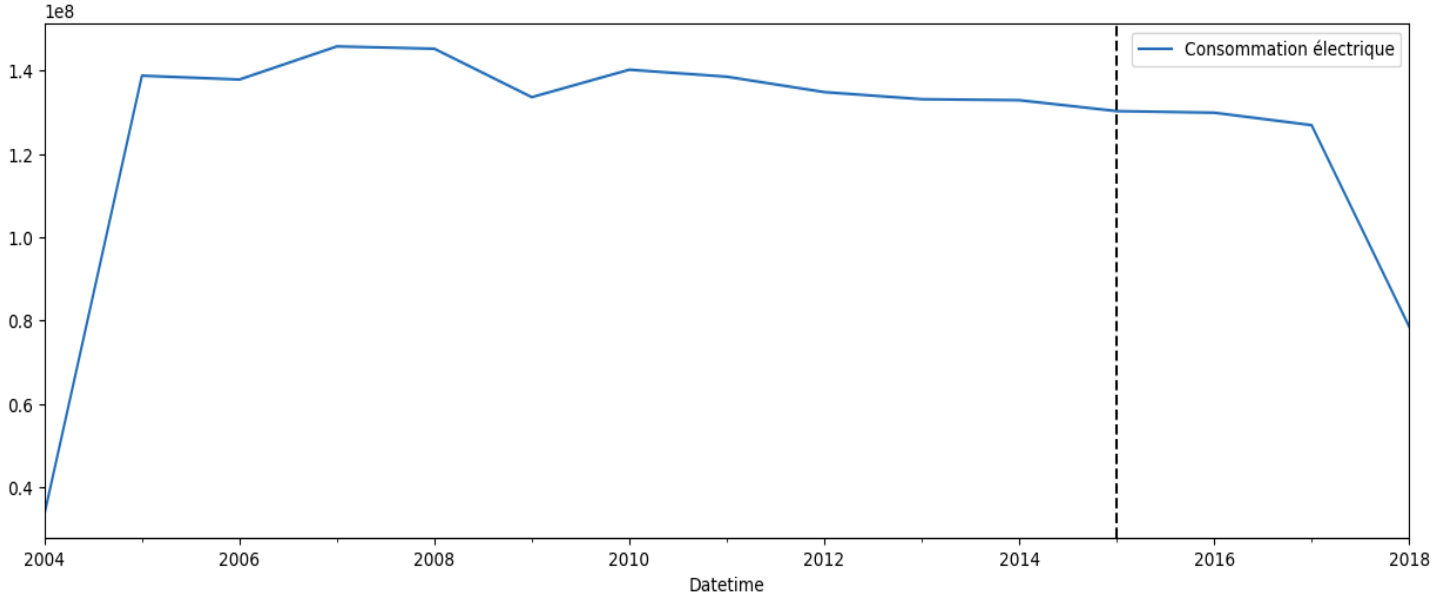
RMSE : 45 819 MW

En moyenne aux Etats Unis, consommation d'un ménage : 10,8 MWh

- **36 655 200 ménages** sont privés d'électricité

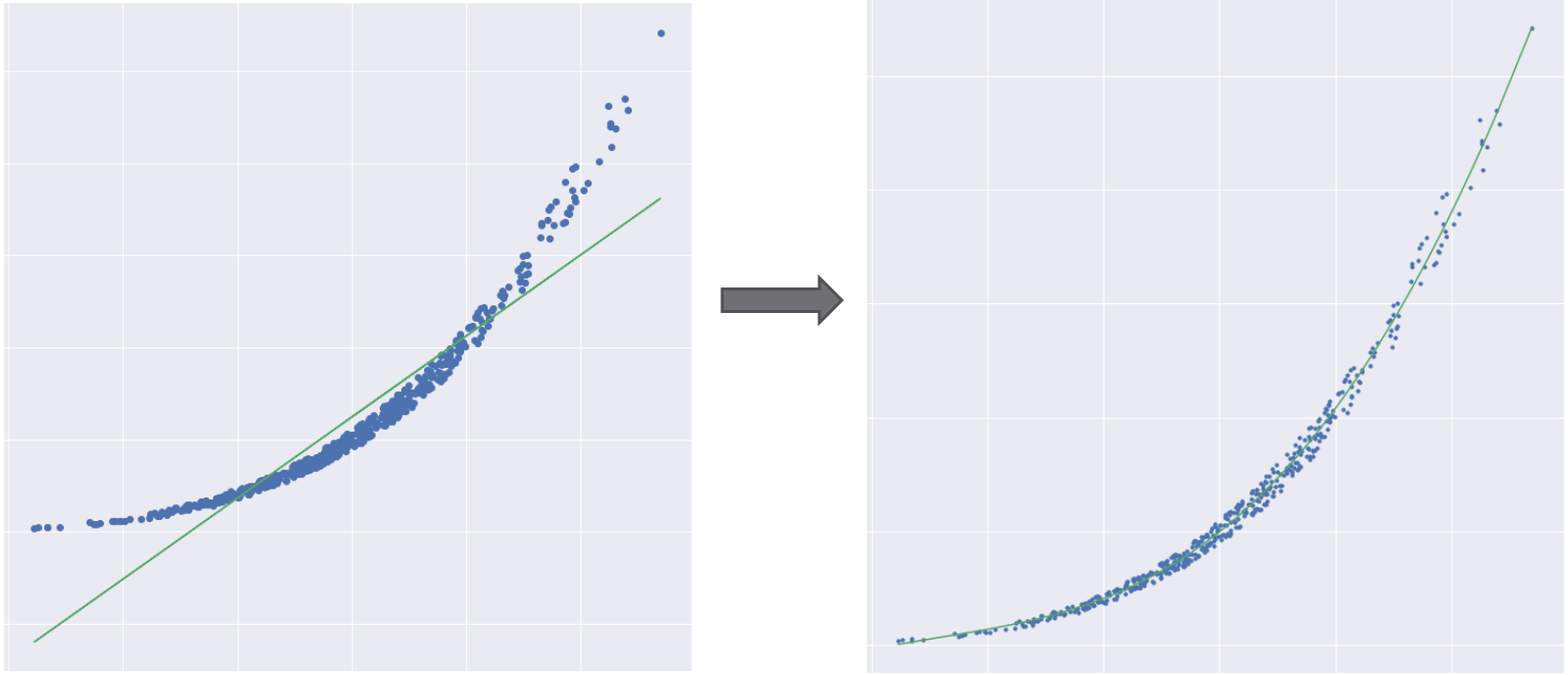
Modèle statistique

Consommation par jour



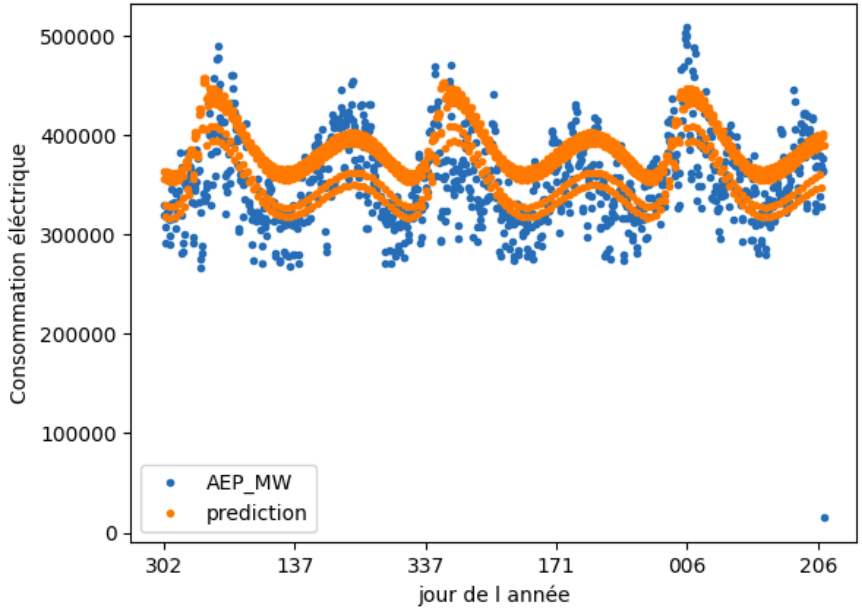
Modèle de Machine Learning

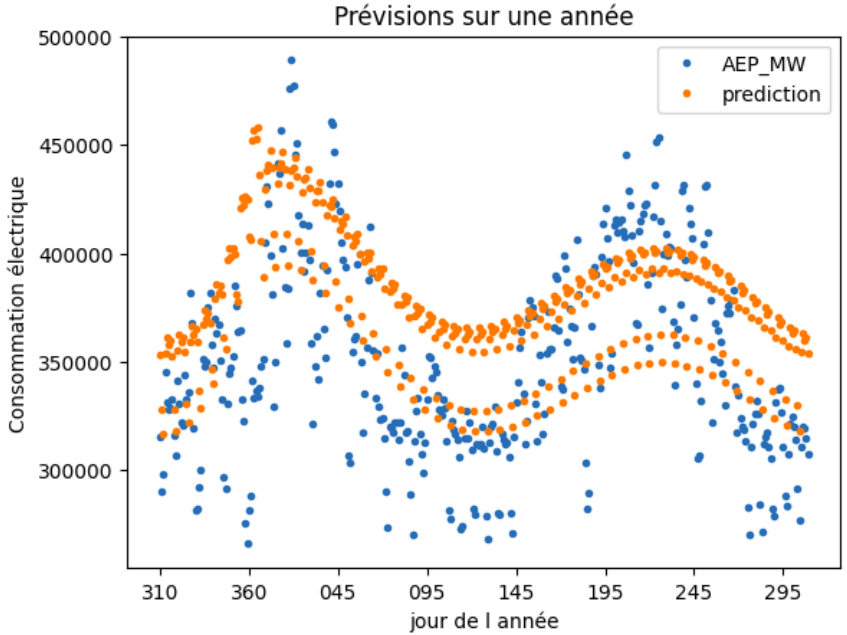
Modèle Polynomial



Modèle de Machine Learning

Modèle Polynomial de degré 5

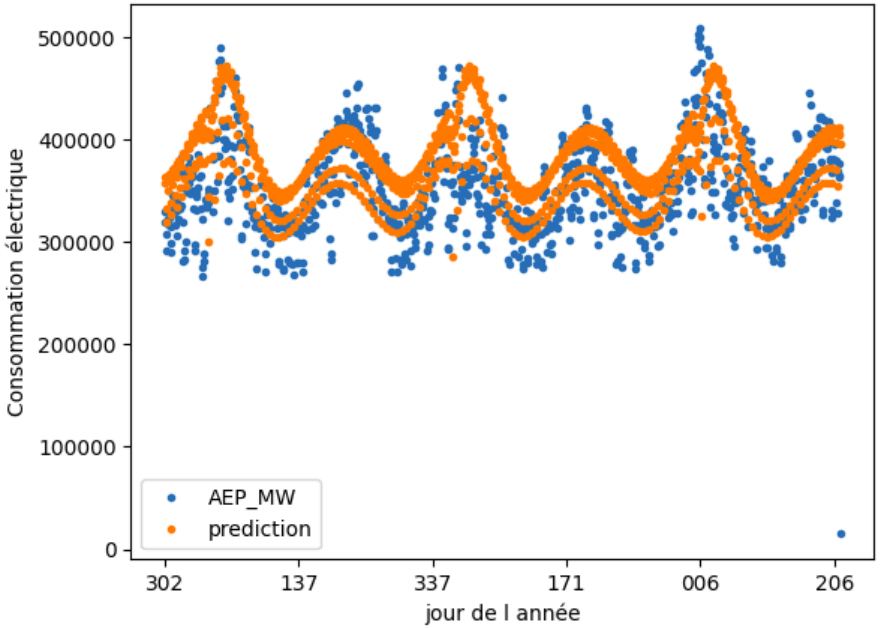


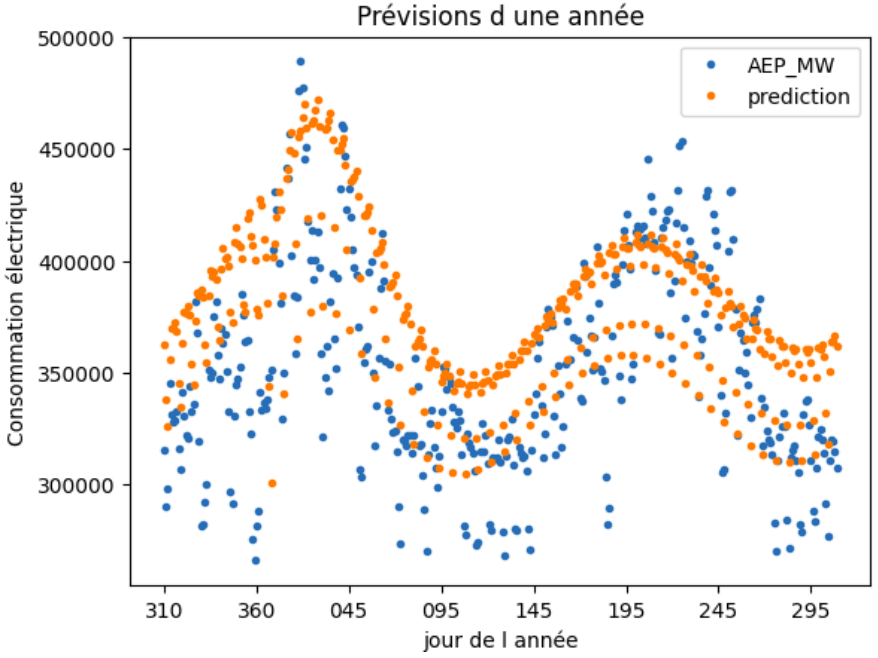


RMSE: 43 975 MW → On passe à 35 180 000 ménages

Modèle de Machine Learning

Modèle Polynomial de degré 6





RMSE : 44 714 MW ⟹ **Sur-apprentissage** : à un degré supplémentaire l'algorithme apprend par cœur les données sur lesquelles il s'est entraîné

Modèle de boosting de gradient (extrême)

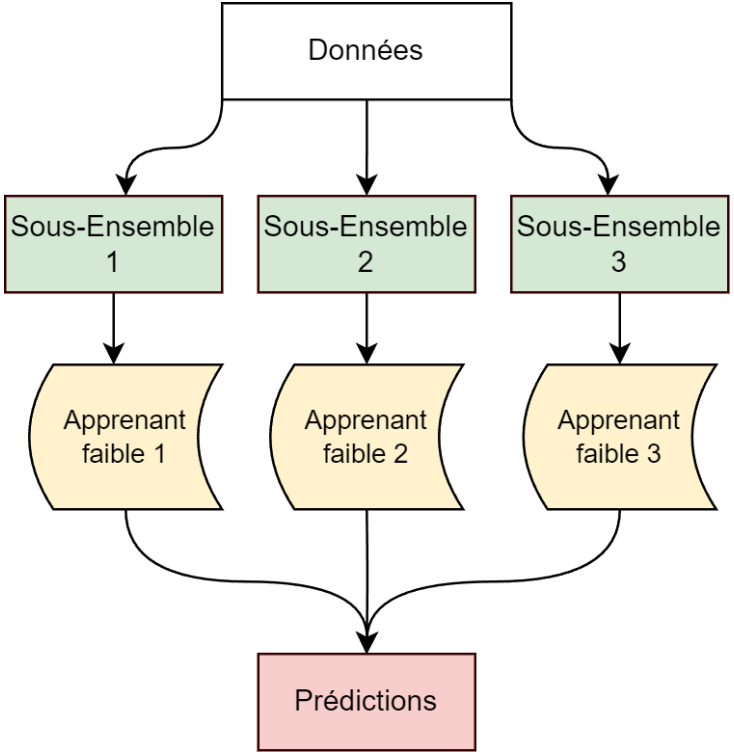
Paramètres supplémentaires

Création de nouveaux paramètres corrélés à la consommation électrique :

- **L'heure**
- **Le jour**
- **La semaine**
- **Le mois**
- **L'année**

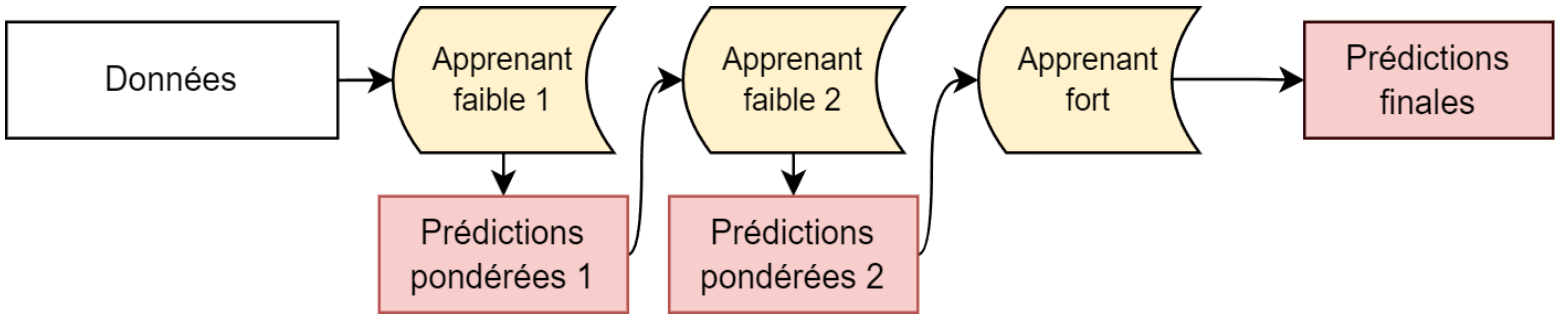
Modèle de boosting de gradient (extrême)

Point Théorique : Apprentissage d'ensemble



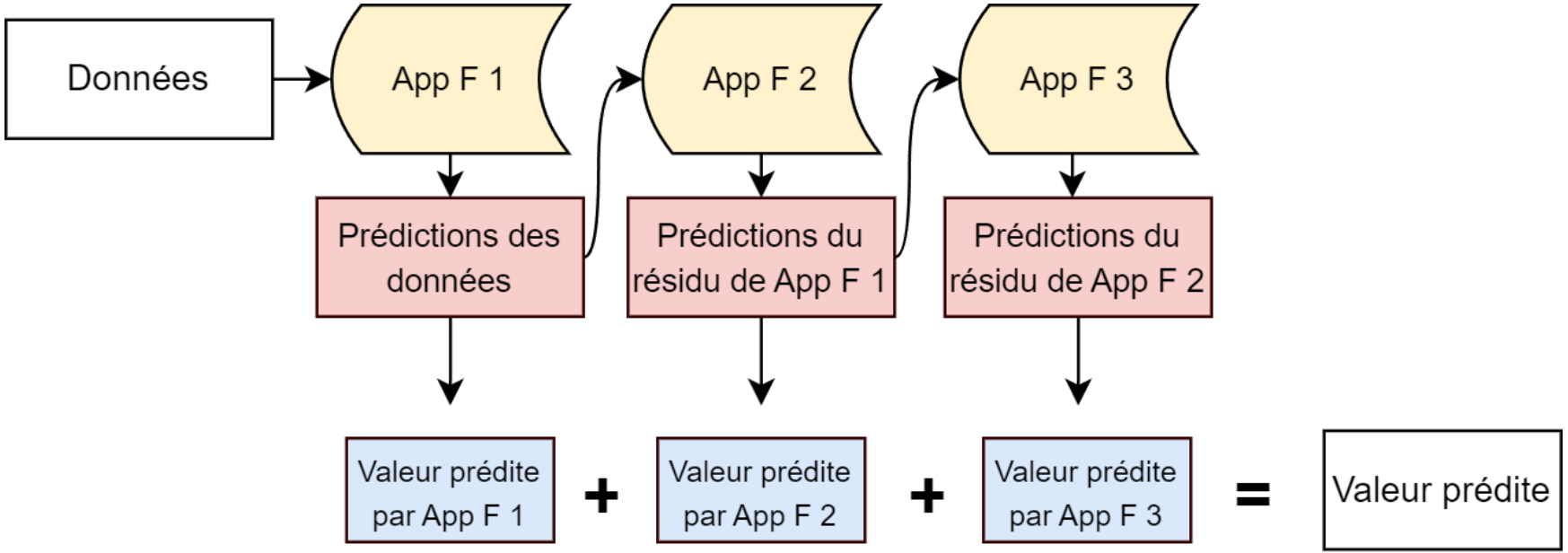
Modèle de boosting de gradient (extrême)

Point Théorique : Boosting



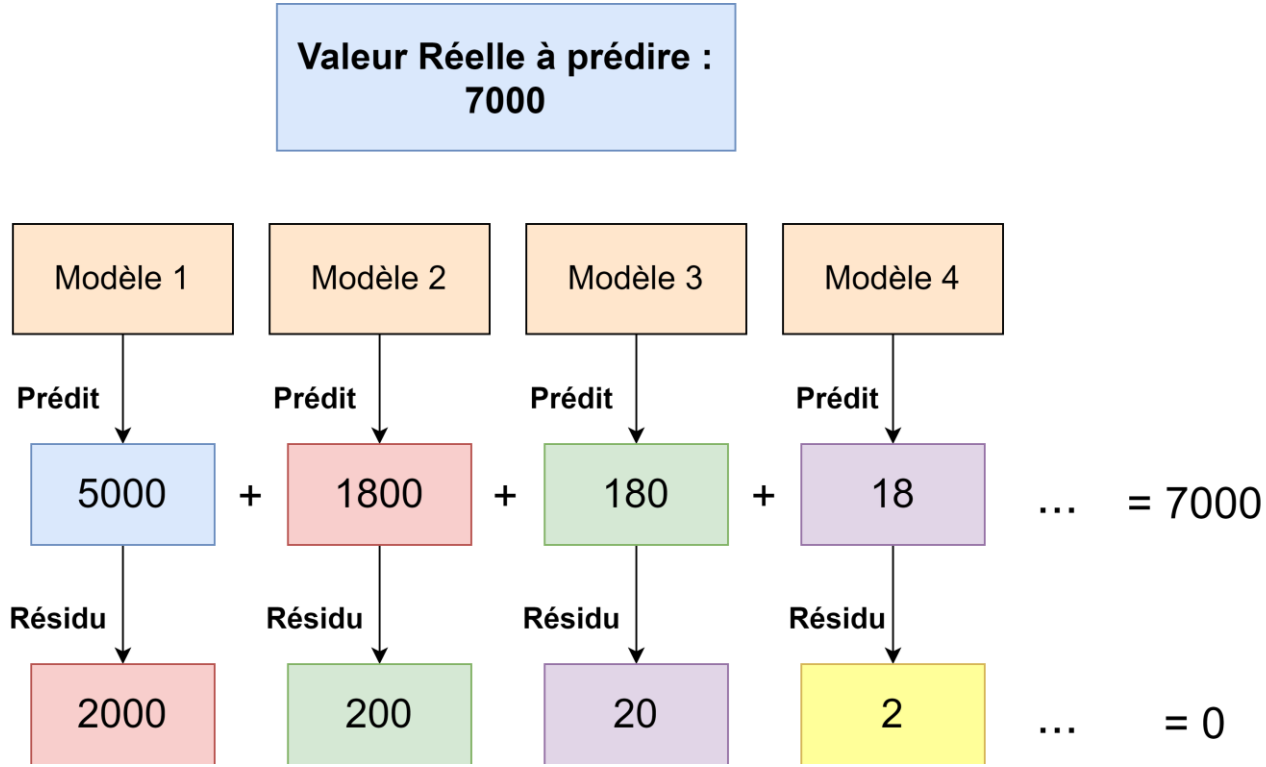
Modèle de boosting de gradient (extrême)

Point Théorique : Boosting de Gradient



Modèle de boosting de gradient (extrême)

Exemple



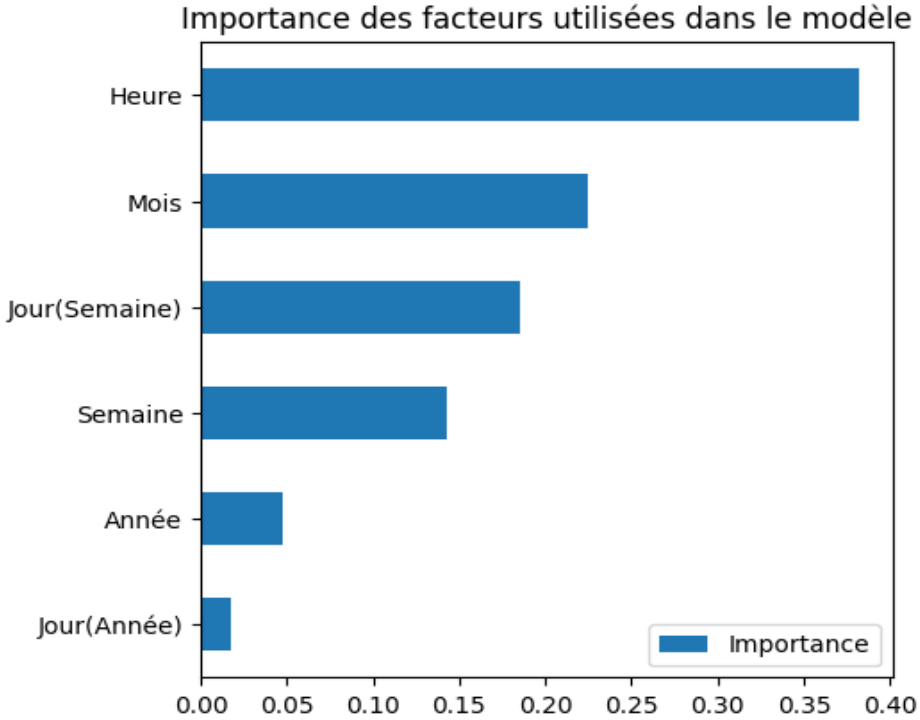
Modèle de boosting de gradient (extrême)

Conditions d'arrêt de l'algorithme

- L'algorithme fait diminuer la RMSE sur le set d'entraînement.
- Parallèlement, le modèle devient de plus en plus précis sur le set de test.
- Arrêt de l'algorithme quand la RMSE sur le set de test augmente.
- Prévient le phénomène d'overfitting.

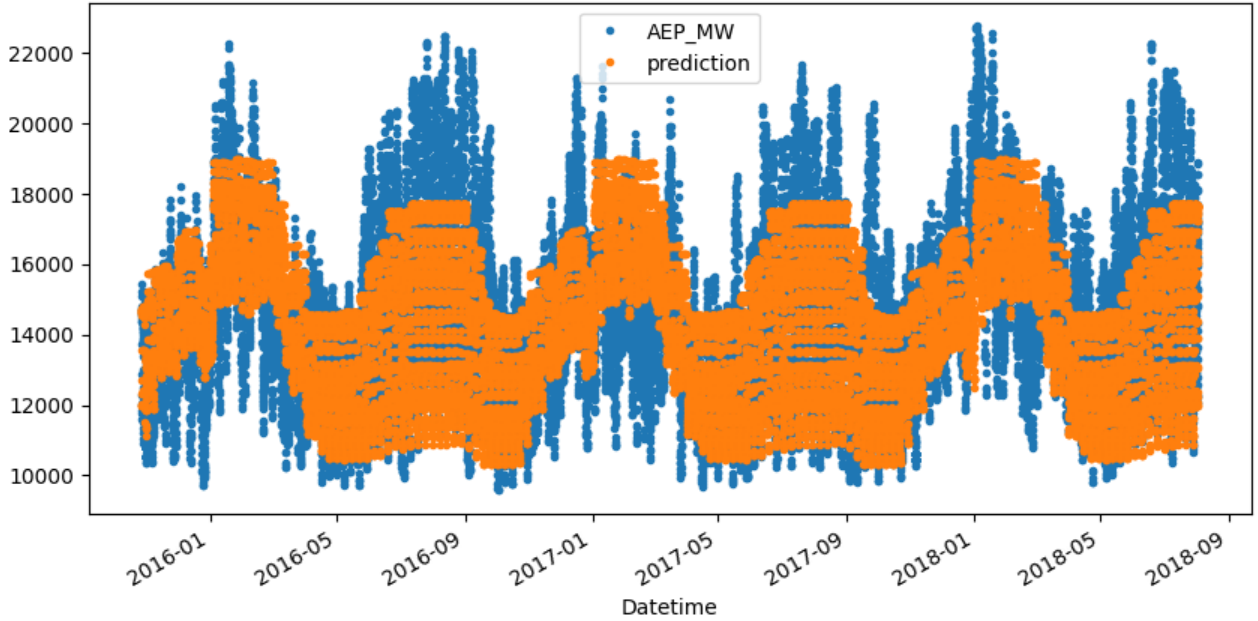
Modèle de boosting de gradient (extrême)

Paramètres dominants



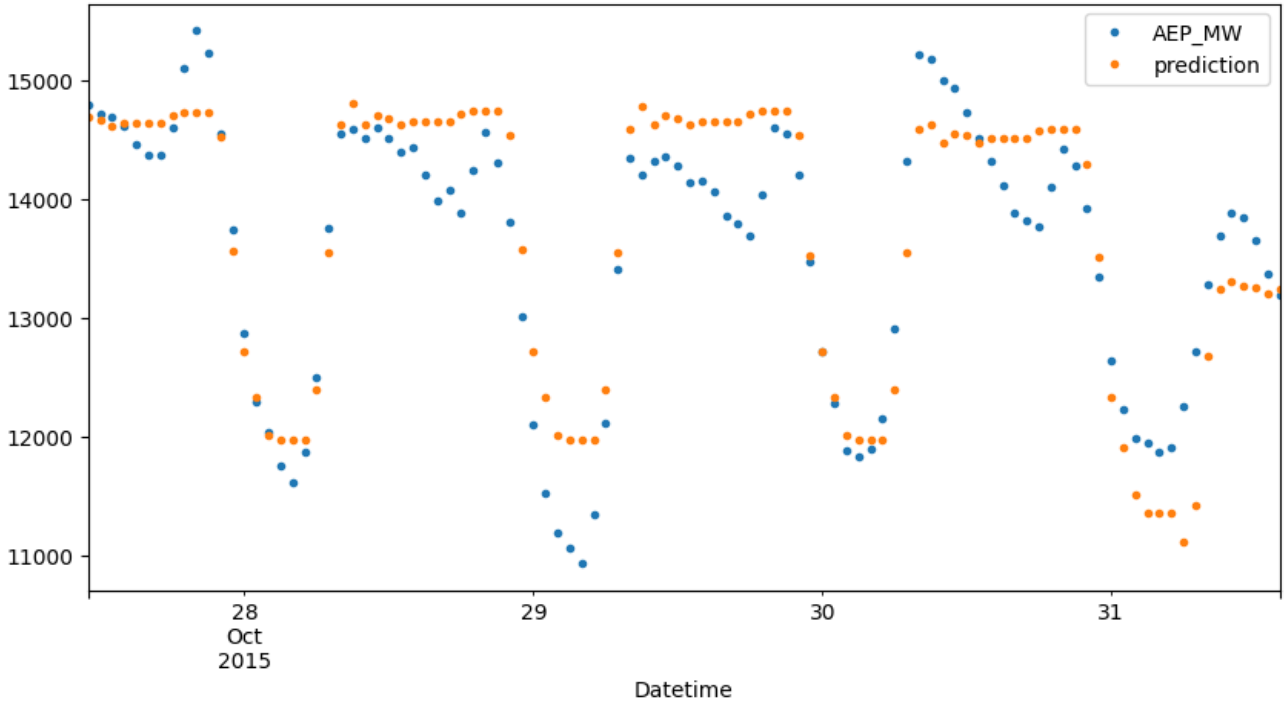
Modèle de boosting de gradient (extrême)

Affichage des prédictions sur le set de test uniquement



Modèle de boosting de gradient (extrême)

Affichage de 100 données prédites



Modèle de boosting de gradient (extrême)

Commentaires

Meilleur modèle prédictif :

- Meilleur rapport Vitesse d'exécution / Précision des prédictions
- Meilleure RMSE : 1616.5 MW

Limite du modèle prédictif :

- Précision des prédictions point par point
- Précision des pics de consommation

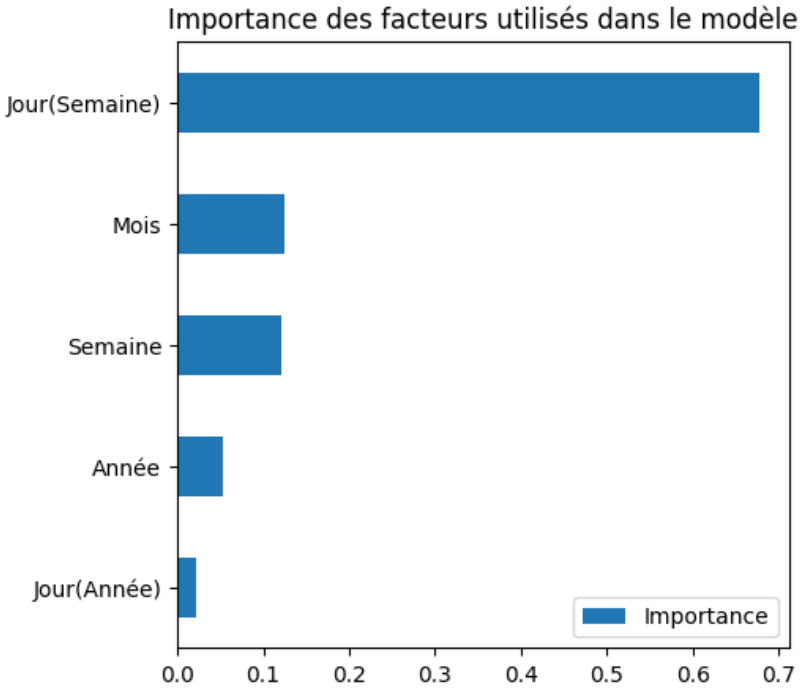
Modèle XGBoost avec température

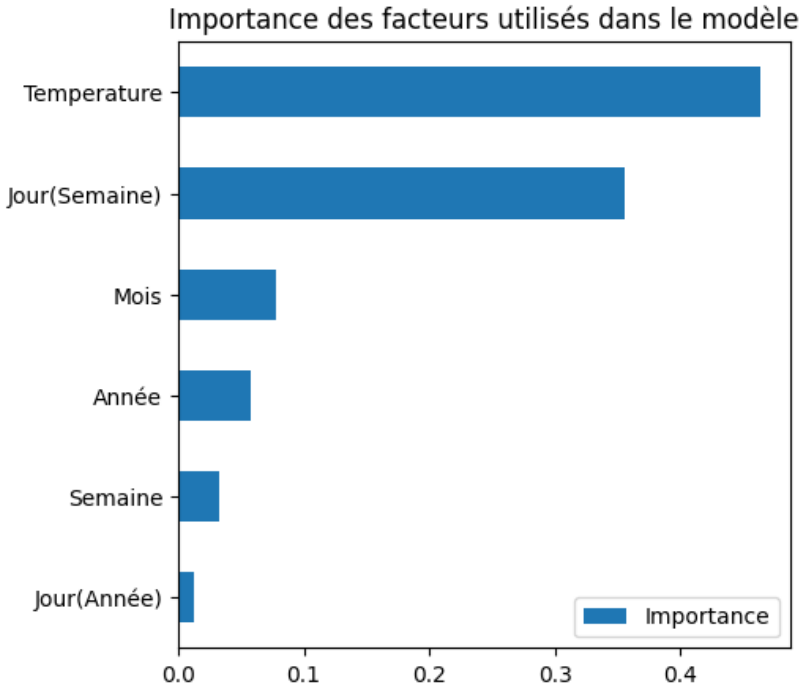
Nouveau choix de base de données

- **13 ans de données :** 1 janvier 2005 au 31 juillet 2018
- **Fréquence:** heure par heure
- **3 colonnes**
- **119 020 lignes**

Modèle XGBoost avec température

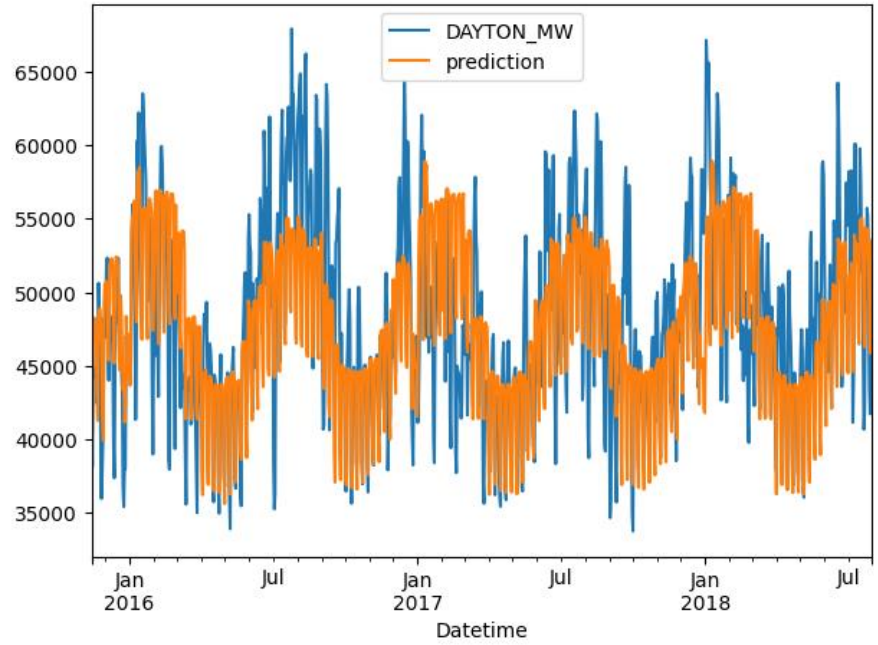
Comparaison de l'importance des paramètres

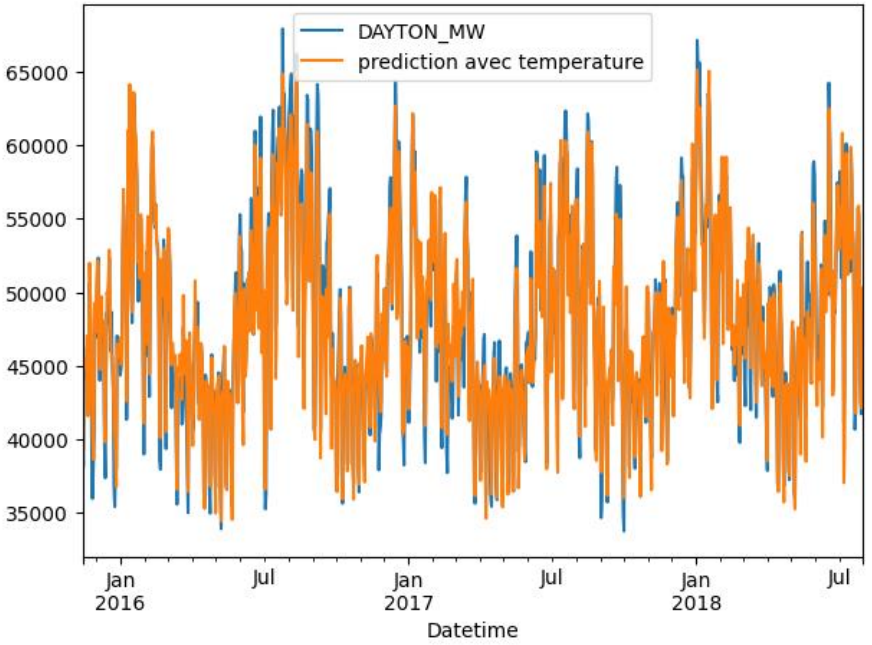




Modèle XGBoost avec température

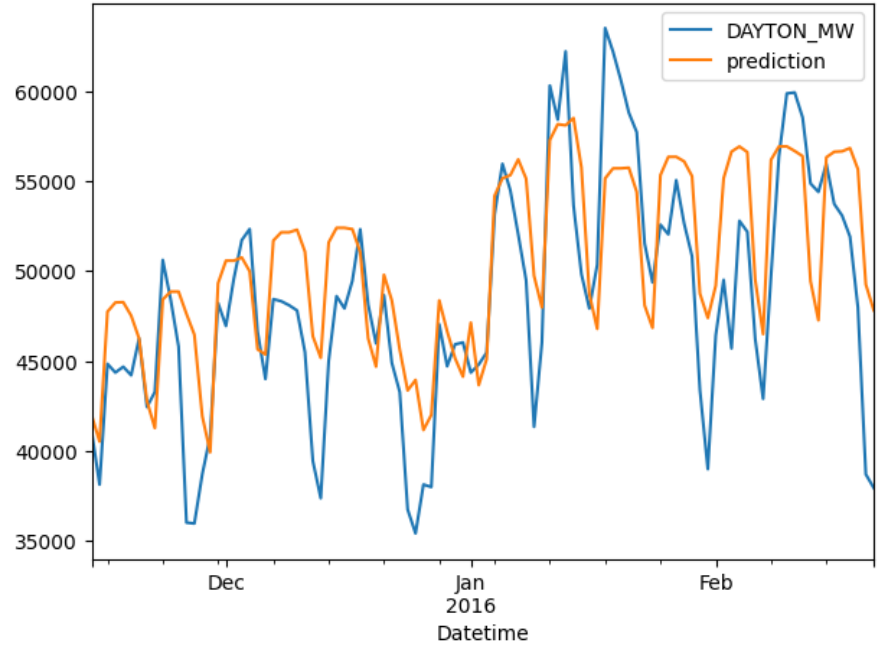
Comparaison des prédictions sur le set de test uniquement

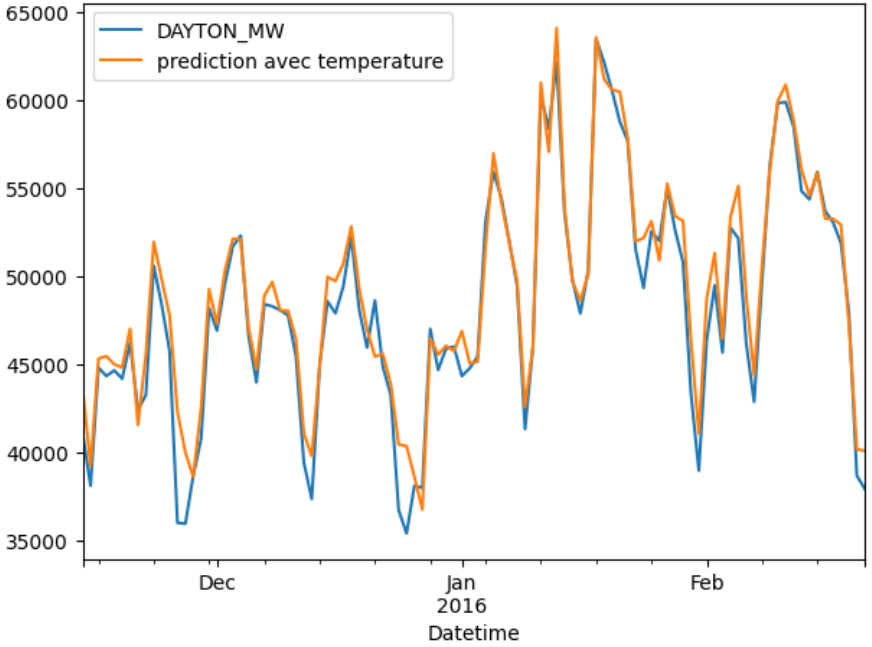




Modèle XGBoost avec température

Comparaison sur 100 données prédites





Modèle XGBoost avec température

Commentaires

Avantages de l'ajout de la température :

- Erreur divisée par 3 passant de 5040 MW à 1695 MW
- On passe donc de 4 032 000 à 1 356 000 ménages
- Plus l'erreur est faible, plus il est difficile de la diminuer
- Prédiction précise des pics

Prochaine étape

- Prévoir les événements particuliers
  - Evènements sportifs
  - Jour particulier en ville (fête de la musique)

- Ajouter de nouveaux paramètres
  - Moyenne des notations DPE des bâtiments dans la ville

Annexe

Base de données

Consommation électrique aux Etats-Unis

```
    #info du dataset
    dataset.info()
    dataset.shape
```

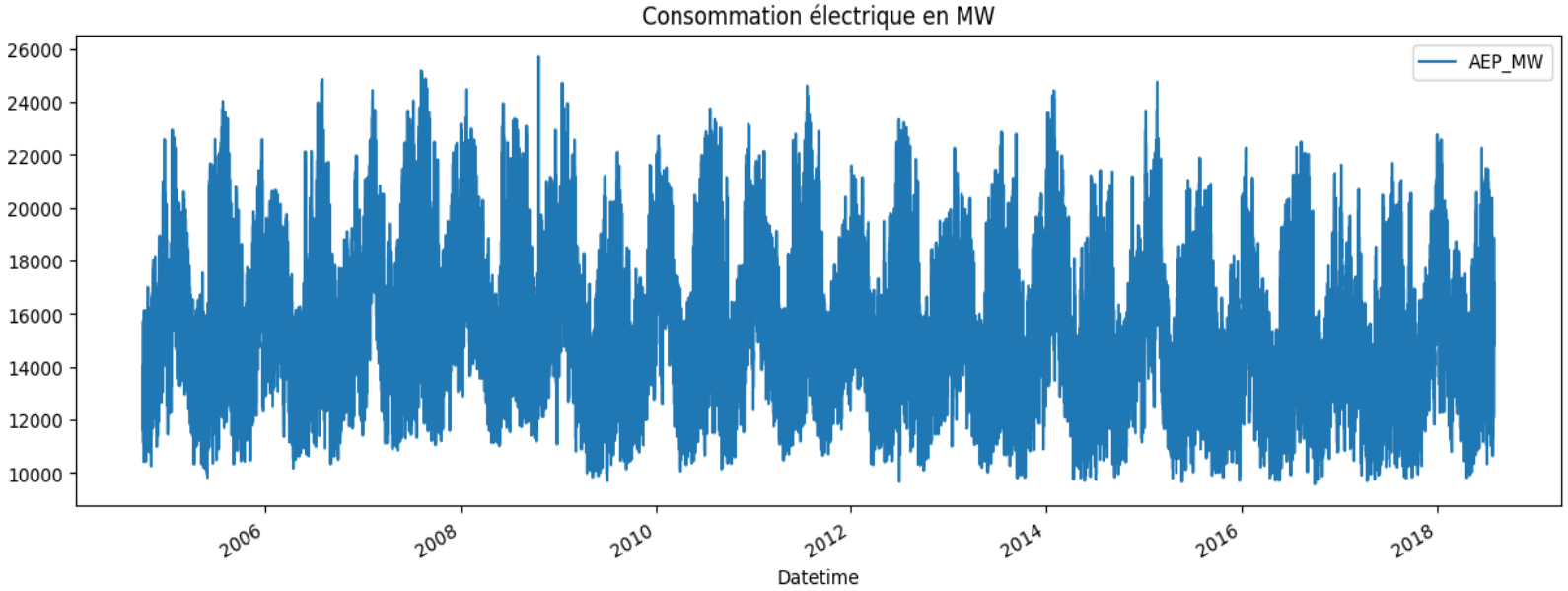
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 121273 entries, 2004-12-31 01:00:00 to 2018-01-02 00:00:00
Data columns (total 1 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   AEP_MW  121273 non-null  float64
dtypes: float64(1)
memory usage: 1.9 MB

(121273, 1)
```

Base de données

Affichage des données

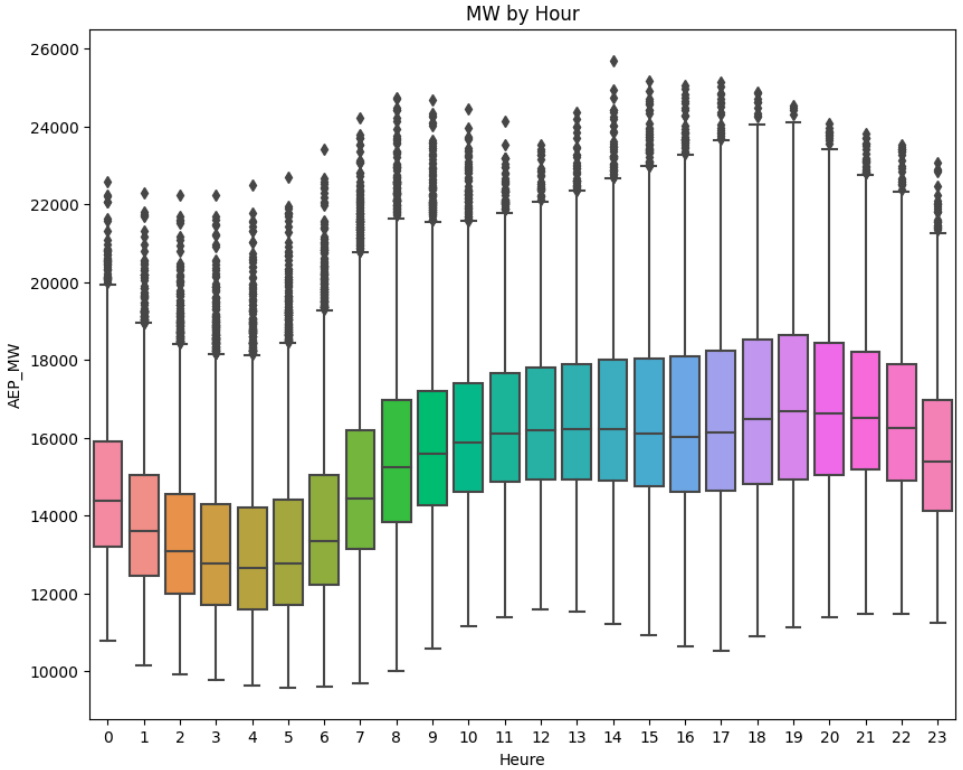
```
# plot des données
dataset.plot(figsize=(15,5),title='Consommation électrique en MW')
```



Base de données

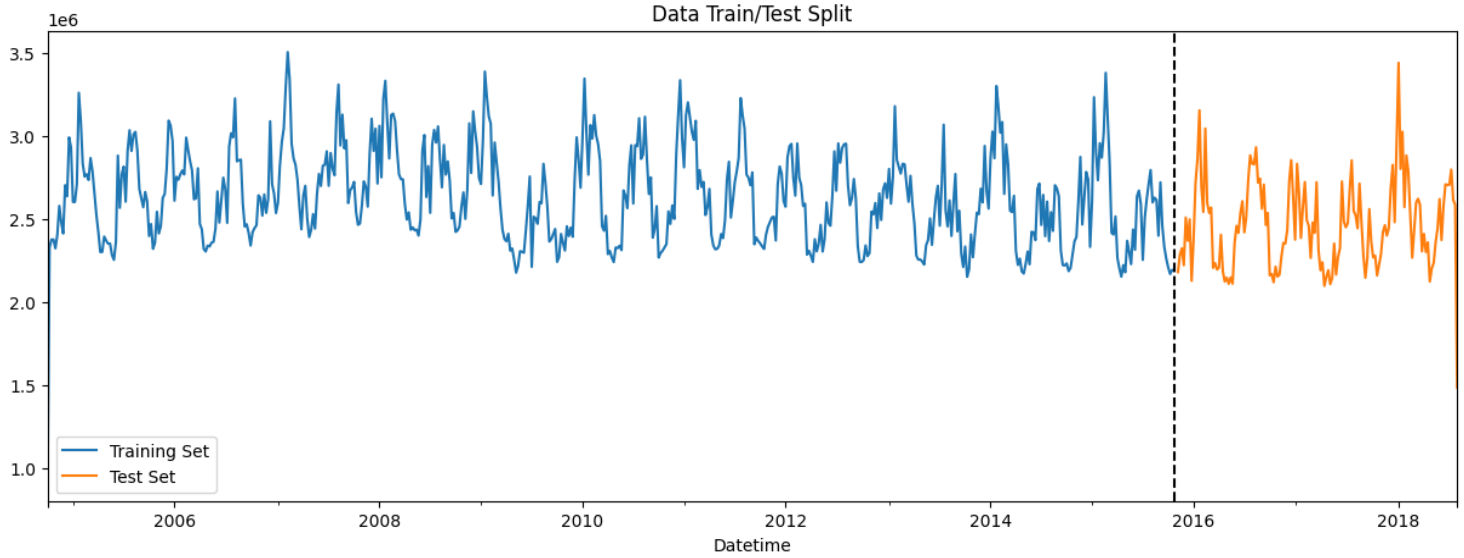
Affichage des données

```
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=ds, x='Heure', y='AEP_MW')
ax.set_title('MW par heure')
plt.show()
```



Partage des données : Train / Test

```
train = daily_data.iloc[:int(nb_lines*0.8)]
test = daily_data.iloc[int(nb_lines*0.8)+1:]
```



Erreur / évaluation des modèles

Utilisation de la bibliothèque Sklearn:

```
from sklearn.metrics import mean_squared_error
```

Modèle statistique

Regroupement par jour

```
daily_groups = dataset.resample('D')
daily_data = daily_groups.sum()
```

```
#numérote les jours de 1 à 7 et de 1 à 365
daily_data["day_of_week"] = daily_data.index.isocalendar().day
daily_data["day_of_year"] = daily_data.index.strftime("%j")
```

Modèle statistique

Consommation par jour

```
#moyenne de la consommation des années précédentes
train_model = train.groupby(by=["day_of_year"]).mean()
train_model = train_model.rename(columns={"AEP_MW":"prediction"})
```

```
#renvoie les prédictions dans une colonne
def predict(df,model):
    return df.merge(model, on ="day_of_year",how="left")
test_predictions_day = predict(test,train_model)
```

Modèle de Machine Learning

Modèle Polynomial de degré 5

```
# fit du modèle de degré 5
test_predictions_p = test

X = train[["day_of_year", "day_of_week"]].values

model = Pipeline([('poly', PolynomialFeatures(degree=5)),
                  ('linear', LinearRegression(fit_intercept=True))])
model.fit(X, train["AEP_MW"].values)
test_predictions_p["prediction"] = model.predict(test[["day_of_year", "day_of_week"]].values)
```

Modèle de Machine Learning

Modèle Polynomial de degré 6

```
# fit du modèle de degré 6
test_predictions_pbis = test

X = train[["day_of_year", "day_of_week"]].values

model = Pipeline([('poly', PolynomialFeatures(degree=6)),
                  ('linear', LinearRegression(fit_intercept=True))])
model.fit(X, train["AEP_MW"].values)
test_predictions_pbis["prediction"] = model.predict(test[["day_of_year", "day_of_week"]].values)
```

Modèle de boosting de gradient (extrême)

Paramètres supplémentaires

```
1  def creation_index_temps(ds):
2      ds['Heure']=ds.index.hour
3      ds['Jour(Semaine)']=ds.index.dayofweek
4      ds['Semaine']=ds.index.week
5      ds['Mois']=ds.index.month
6      ds['Année']=ds.index.year
7      ds['Jour(Année)']=ds.index.day
8      return ds
9
10 ds = creation_index_temps(dataset)
```

Modèle de boosting de gradient (extrême)

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).

```
ENTREES = ['Heure', 'Jour(Semaine)', 'Semaine', 'Mois', 'Année', 'Jour(Année)']
SORTIE = ['AEP_MW']
```

```
X_train = train[ENTREES]
Y_train = train[SORTIE]

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

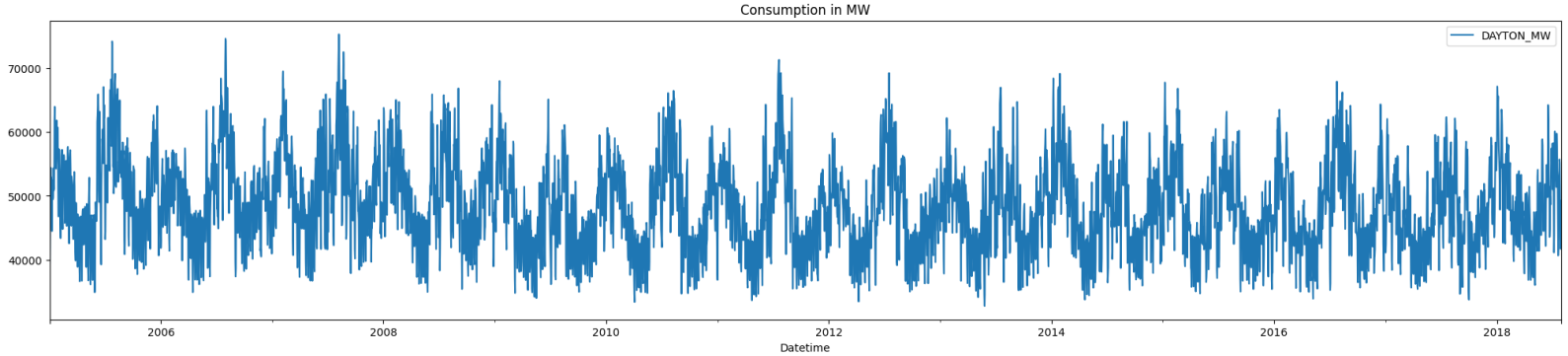
Modèle de boosting de gradient (extrême)

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

```
reg = xgb.XGBRegressor(
    n_estimators=100000,         # Hauteur maximale de l'arbre de décision créé lors du boosting
    early_stopping_rounds=100,   # 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 100 itérations)
```

Modèle XGBoost avec température

Nouveau choix de base de données

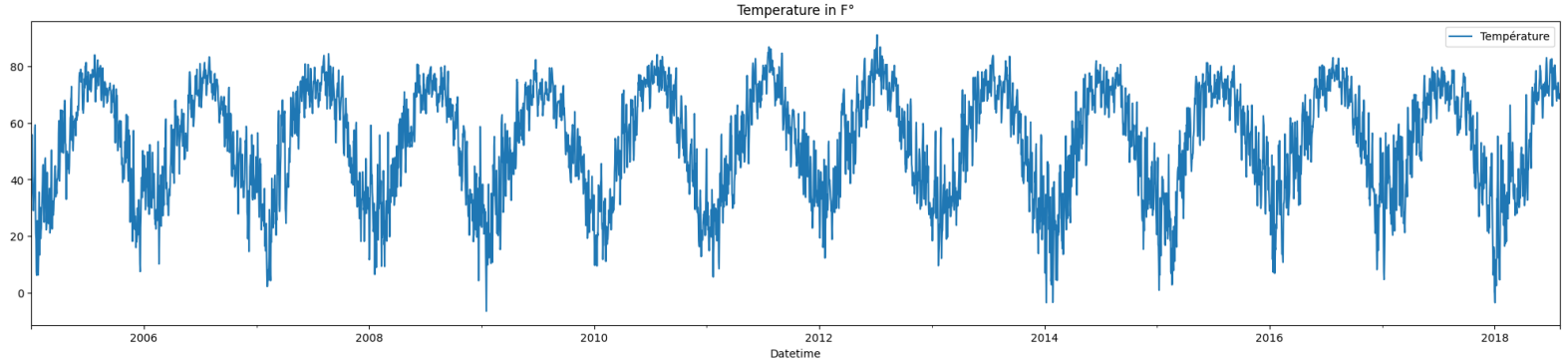



```
1  # plot des données
2  dataset.plot(figsize=(15,5),title='Energy use in MW')
```

✓ 0.7s Python

Modèle XGBoost avec température

Jointure des bases de données



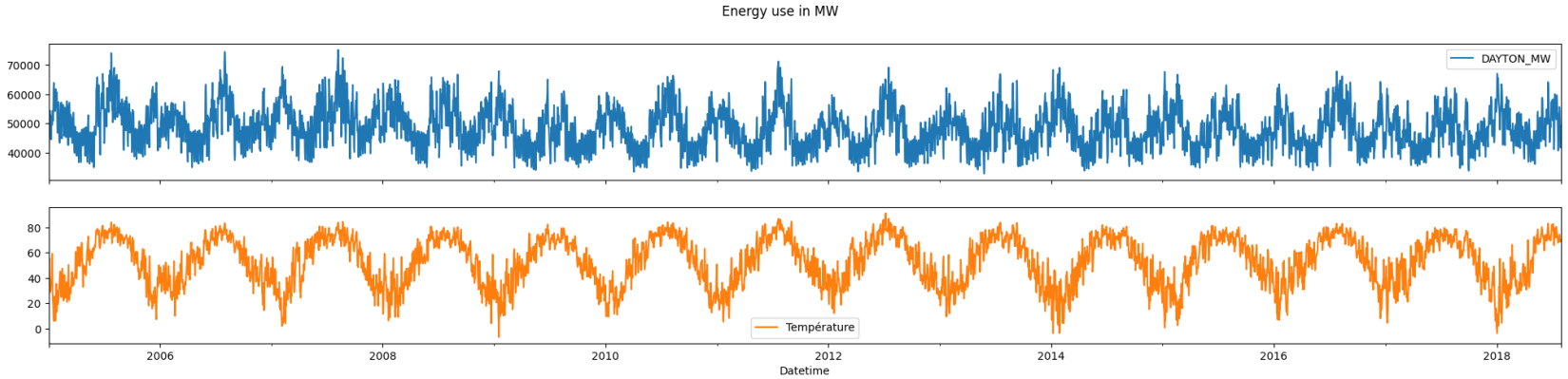

```
1  # plot des données
2  dataset1.plot(figsize=(15,5),title='Energy use in MW')
```

✓ 0.7s Python

Modèle XGBoost avec température

Jointure des bases de données

```
1  dataset_final=dataset_daily.join(dataset1,on='Datetime')
```



Modèles XGBoost

Traçage des données présentes dans le set de données "Test" ainsi que les prédictions estimées par le modèle.

```
test['prediction'] = reg.predict(X_test)
dataset_final = dataset_final.merge(test[['prediction']], how='left', left_index=True, right_index=True)
```

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