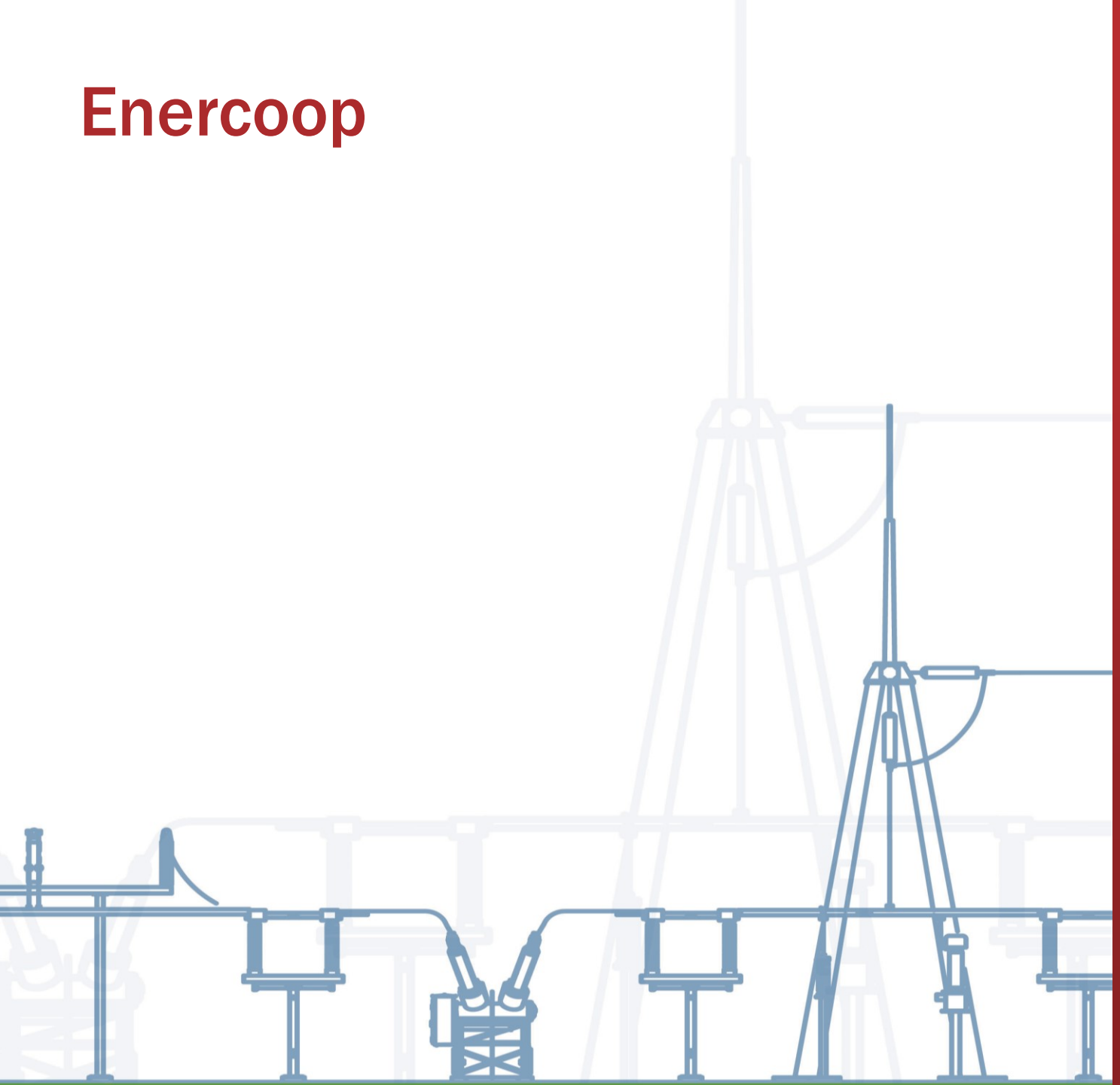
Enercoop



Prédire la demande en électricité.

Projet 9 – DA – Marc SELLAM - 01/2021

Sommaire :

- Page 1 Mission.
- Page 2 Les données fournies.
- Page 4 Préparation des données.
- Page 6 Correction des données de consommation mensuelles de l'effet température (dues au chauffage électrique).
- Page 7 Désaisonnalisation de la consommation obtenue après correction, grâce aux moyennes mobiles.
- Page 9 Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode de Holt Winters (lissage exponentiel).
- Page 10 Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode SARIMA sur la série temporelle.
- Page 15 Comparatif de tous les modèles.
- Page 16 Conclusion.

Mission

- Enercoop est une société coopérative qui s'est développée grâce à la libéralisation du marché de l'électricité en France.
- La demande en électricité des utilisateurs varie au cours du temps et dépend de paramètres comme la météo (température, luminosité, etc.) .
- Tout le challenge est de mettre en adéquation l'offre et la demande, en créant un model de prévision de la consommation électrique totale.

Les données fournies

- Téléchargement des données mensuelles de consommation totale d'électricité en énergie sur https://www.rte-france.com/eco2mix/telecharger-les-indicateurs.
  - Consommation totale ile de France en GWh , données mensuelles

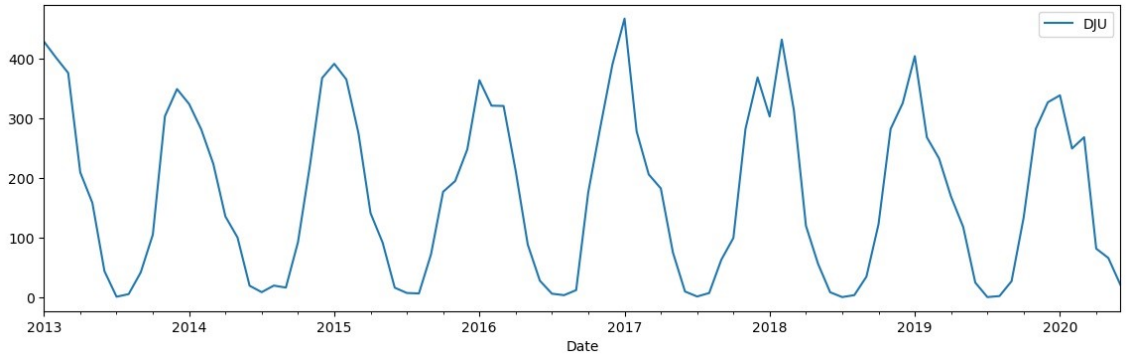
    (unité de mesure d'énergie qui correspond à la puissance d'un gigawatt actif pendant une heure)

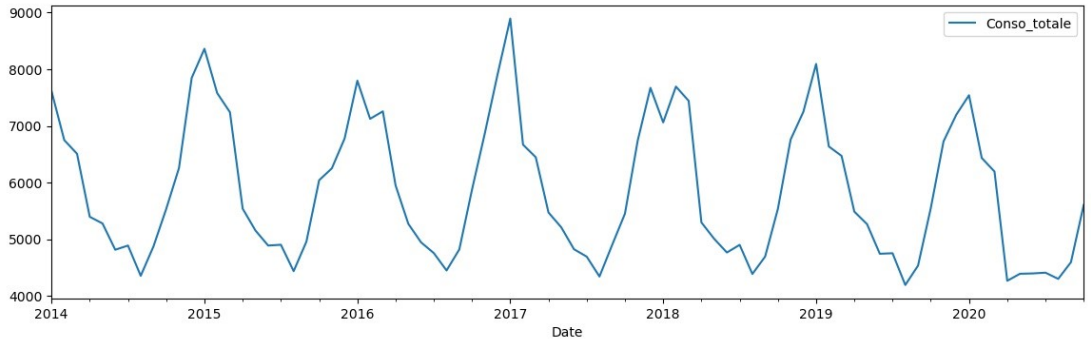
- Données météo utilisées pour corriger les données de l'effet température téléchargées sur https://cegibat.grdf.fr/simulateur/calcul-dju.
  - Dju paris Montsouris, données mensuelles

    (degré Jour Unifié est l'écart entre la température d'une journée donnée et un seuil de température établi à 18 °C)

Les données fournies

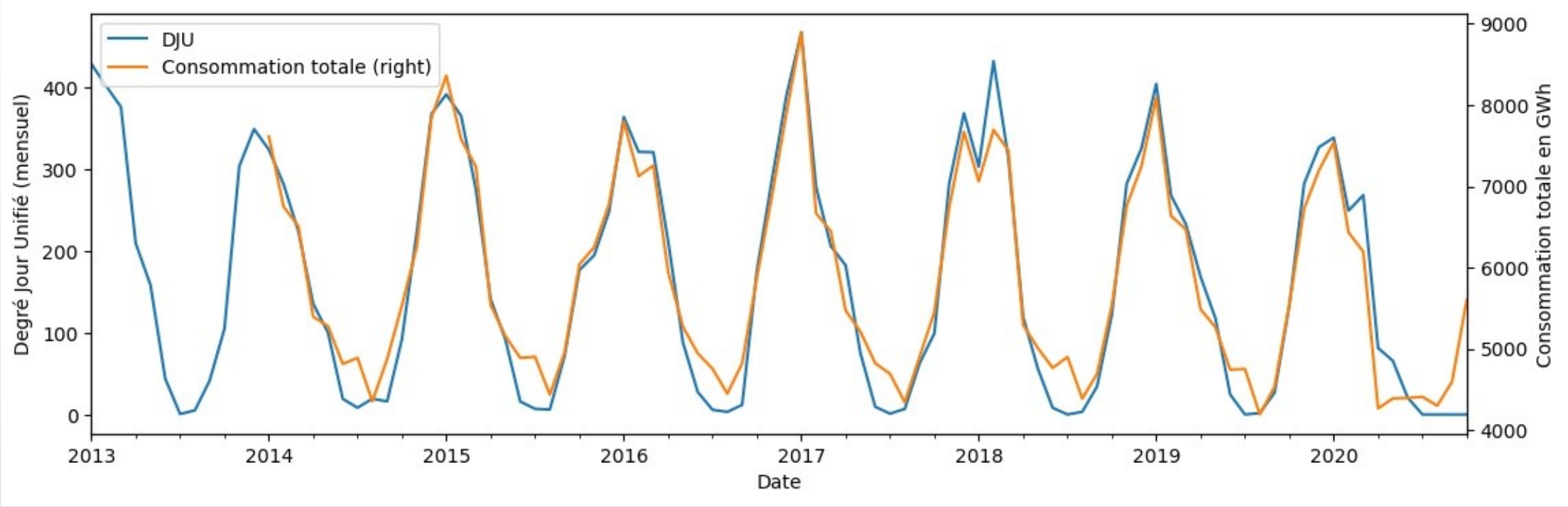
	Année	JAN	FÉV	MAR	AVR	MAI	JUN	JUI	AOÛ	SEP	OCT	NOV	DÉC	Total
11	2020	339	249.6	268.6	81.4	65.7	20.6	0	0	0	0	0	0	1024.9
12	2019	404.9	268.3	233.1	168.5	117.9	24.4	0	1.7	26.7	133.7	282.6	327.3	1989
13	2018	303.4	432.6	314.3	119.7	55.9	8.1	0	3.3	34.3	122.4	282.5	325.9	2002.2
14	2017	467.9	278.4	206.1	182.6	75	9.4	1	6.8	62.6	99.4	282.6	369	2040.6
15	2016	364.4	321.6	321.1	212.1	88.1	27.5	5.7	3.2	11.7	176	285.6	390.8	2207.3
16	2015	392	365.7	275.5	141.1	91.5	15.8	6.9	6.1	71.9	176.9	195	248.1	1986.2
17	2014	324.4	281.9	223.9	135.5	100.2	19.1	8.3	19.3	16	92.3	222.6	368.2	1811.5
18	2013	429.2	402.2	376.6	209.5	158.4	43.6	0.6	5	41.5	105	303.9	349.5	2424.8
19	2012	336	435.9	201.9	230.3	83.3	35	12.4	2.4	58	154.6	296.2	345.9	2191.5
20	2011	392	304.8	243.1	77.6	43.4	31.4	15	11.9	23.2	127.6	226.6	312.7	1809
21	2010	499.2	371.4	294.5	165.3	140.9	22.6	0	11.1	52.3	172.2	310	512	2551.1
22	2009	486.8	365.7	293.2	135.1	82.2	39.8	3.1	0.9	26.9	149.6	224.7	411.8	2219.7





	Date	Conso_totale
19	2013-01	0
32	2013-02	0
45	2013-03	0
58	2013-04	0
71	2013-05	0
...	...	...
1176	2020-06	4397
1189	2020-07	4410
1202	2020-08	4301
1215	2020-09	4595
1228	2020-10	5605

Préparation des données



- Apres concaténation des données :
  - 6 années de données exploitables.
  - Similitude des 2 courbes.

Préparation des données

```
#data_tr =  data_i de 2014 à 2018, annees 'd'entrainement'
data_tr = data_i.loc['2014':'2018'].copy()
#data_te =  data_i de 2019, annee 'test'
data_te = data_i.loc['2019'].copy()
```

- 6 années :

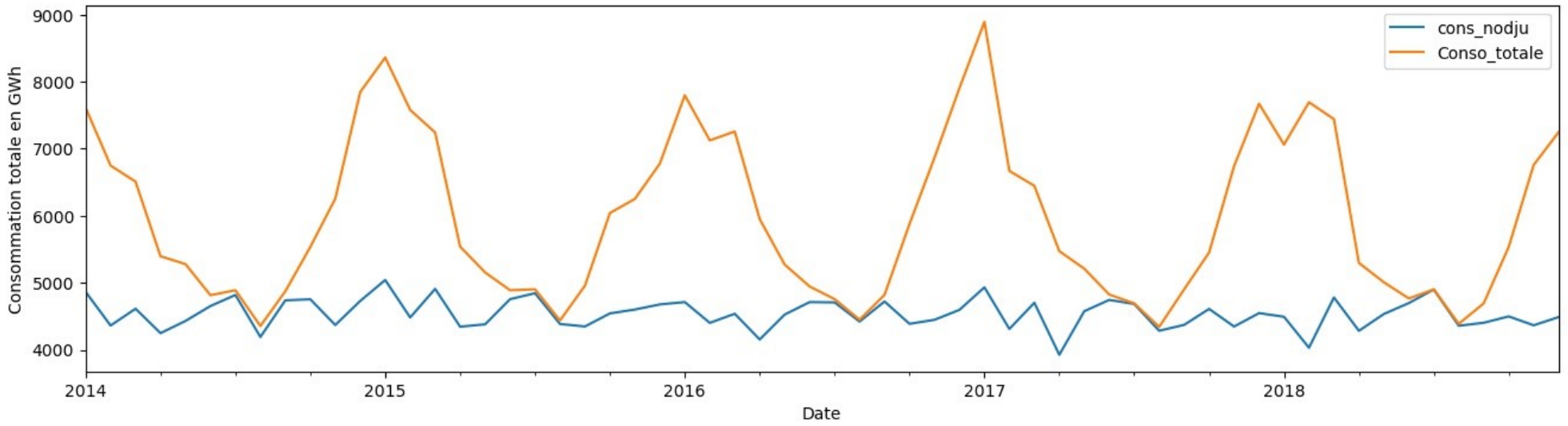
les cinq premières pour l'entrainement

la sixième pour vérifier la qualité des prédictions.

Date	DJU	Conso_totale
2014-01-01	324.4	7612
2014-02-01	281.9	6749
2014-03-01	223.9	6509
2014-04-01	135.5	5396
2014-05-01	100.2	5279
...	...	...
2018-08-01	3.3	4387
2018-09-01	34.3	4694
2018-10-01	122.4	5535
2018-11-01	282.5	6758
2018-12-01	325.9	7248

Date	DJU	Conso_totale
2019-01-01	404.9	8093
2019-02-01	268.3	6637
2019-03-01	233.1	6471
2019-04-01	168.5	5487
2019-05-01	117.9	5266
...	...	...
2019-08-01	1.7	4193
2019-09-01	26.7	4535
2019-10-01	133.7	5549
2019-11-01	282.6	6726
2019-12-01	327.3	7197

Correction des données de consommation mensuelles de l'effet température (dues au chauffage électrique)

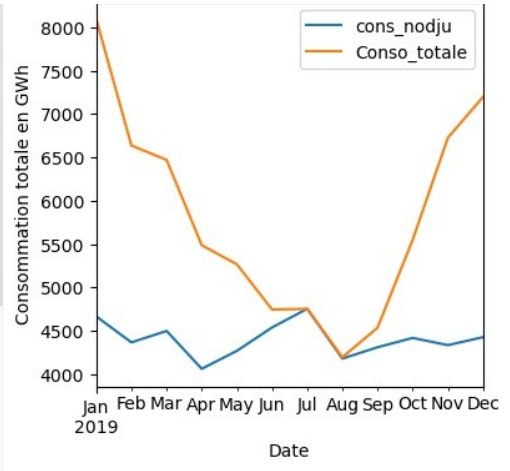



```
x = data_tr[['DJU']].copy()
y = data_tr[['Conso_totale']].copy()
```

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

reg = smf.ols('y~x', data=data_tr).fit()
```

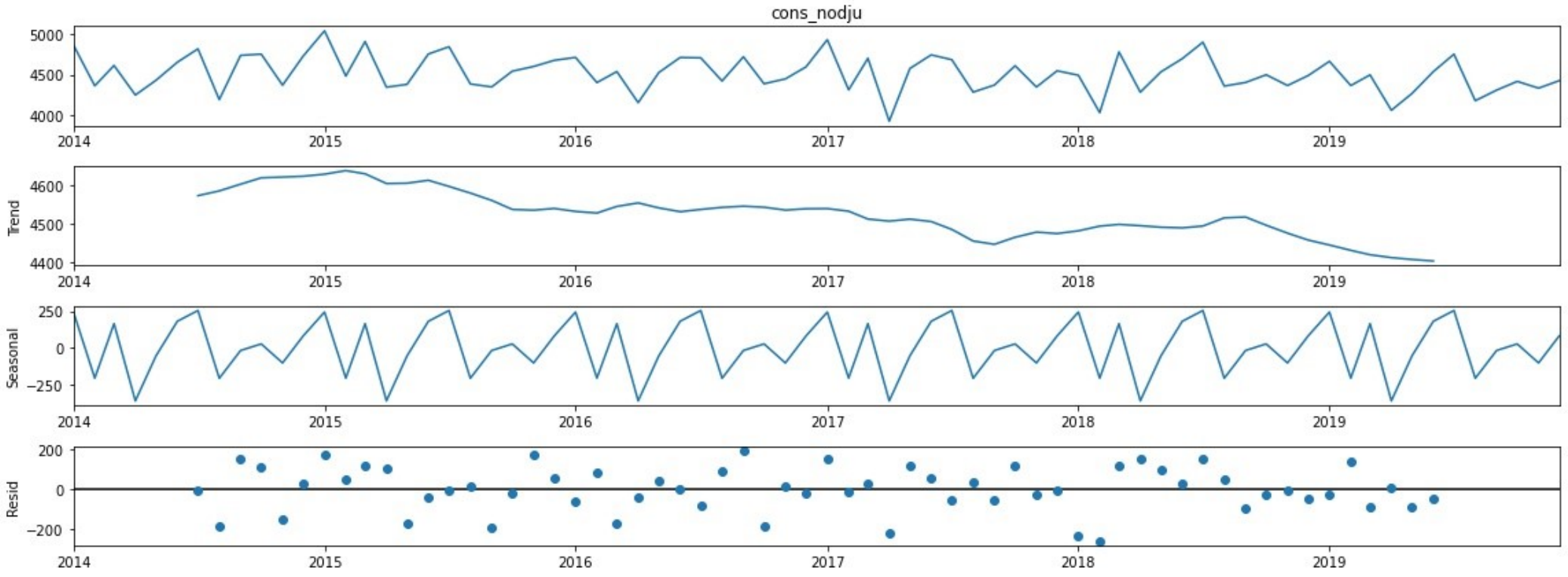
```
coef = reg.params[1]
data_tr['co_dju'] = data_tr['DJU'] * coef
data_tr['cons_nodju'] = data_tr['Conso_totale'] - data_tr['co_dju']
```

- Correction des 6 années.



Désaisonnalisation de la consommation obtenue après correction, grâce aux moyennes mobiles.

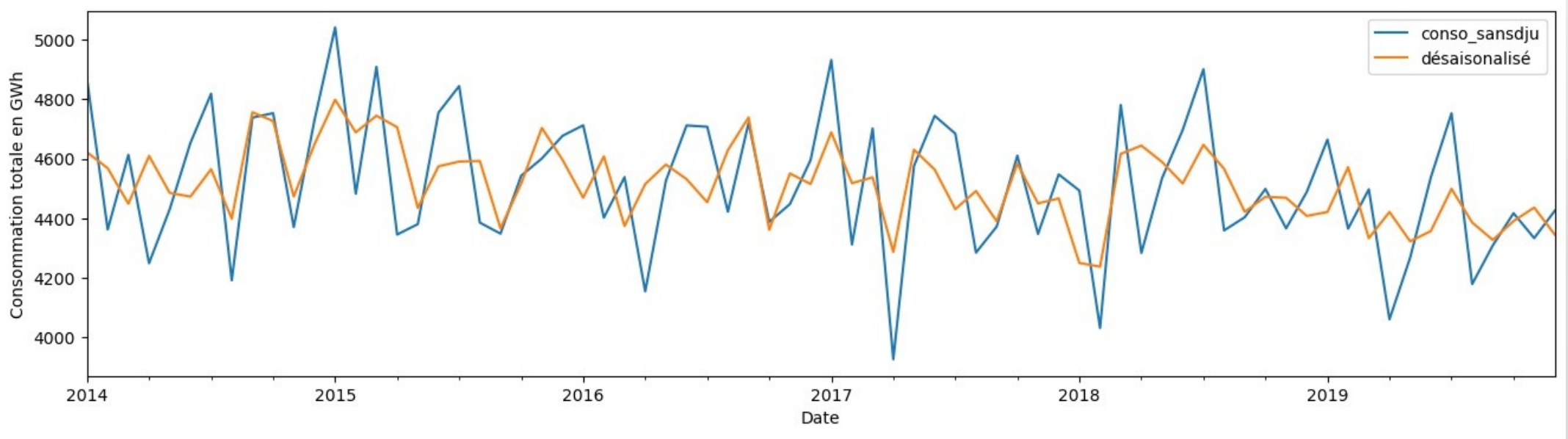
```
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]=[13,4]
decomp_x = seasonal_decompose(data_all['cons_nodju'],  model='additive', period = 12)
fig = decomp_x.plot()
plt.savefig("2_0.jpg",bbox_inches ="tight")
plt.show()
```



Désaisonnalisation de la consommation obtenue après correction, grâce aux moyennes mobiles.

```
#desaisonalisation :
data_all['cons_nodjusea'] = data_all['cons_nodju'] - decomp_x.seasonal
data_all['cons_nodju'].plot(label='conso_sansdju',figsize=(16,4))
data_all['cons_nodjusea'].plot(label = 'désaisonalisé')
plt.legend()
plt.ylabel("Consommation totale en GWh",fontsize=10)
plt.savefig("2_1.jpg",bbox_inches ="tight")
```

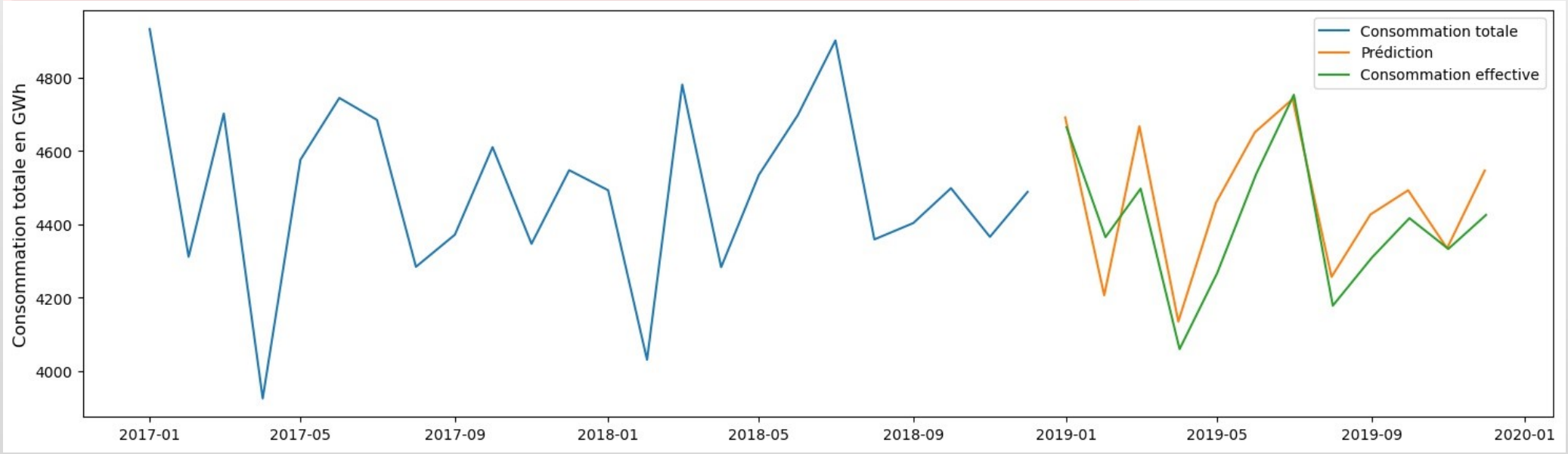
- Désaisonnalisation appliquée aux 6 années



Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode de Holt Winters (lissage exponentiel).

```
y = data_tr[['cons_nodju']].copy()
```

```
from statsmodels.tsa.api import ExponentialSmoothing
hw = ExponentialSmoothing(np.asarray(y), seasonal_periods=12, trend='add', seasonal='add').fit()
hw_pred = hw.forecast(12)
plt.figure(figsize=(18,5))
plt.plot(y.loc['2017':], label='Consommation totale')
plt.plot(pd.date_range(y.index[len(y)-1], periods=12, freq='M'), hw_pred, label='Prédiction')
plt.plot(data_te['cons_nodju'], label='Consommation effective')
plt.ylabel("Consommation totale en GWh",fontsize=12)
plt.legend()
plt.savefig("3_1.jpg",bbox_inches ="tight")
plt.show()
```



Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode SARIMA sur la série temporelle.

Vérification de stationnarités et différenciations

Sur les données de test :

Consommation non stationnaire (1)

Apres différenciation :

Consommation avec 1 décalage(2) : stationnaire.

Consommation avec 2 décalage(3) : stationnaire.

Consommation avec 12 décalage(4) : stationnaire.

```
adf_check(data_tr['cons_nodju'])
```

(1)

```
Test de Dickey_fuller Augmenté (adf)
ADF Test Statistic:-0.9735442358946739
p-value:0.762790741186066
#lags used:11
Number of observations used:48
Faible évidence contre l'hypothèse nulle
Echoue à rejetter l'hypothèse nulle
Les données ont une racine unitaire et sont donc non stationnaires
```

```
#data_tr['First_difference'] = data_tr['cons_nodju'] - data_tr['cons_nodju'].shift(1)
adf_check(data_tr['First_difference'].dropna())#perte 1ere valeur
```

(2)

```
Test de Dickey_fuller Augmenté (adf)
ADF Test Statistic:-4.428884196362779
p-value:0.00026377732851989177
#lags used:10
Number of observations used:48
Forte preuve contre l'hypothèse nulle
Rejette l'hypothèse nulle
Les données n'ont pas de racine unitaire et sont donc stationnaires
```

```
#data_tr['Seconde_difference'] = data_tr['First_difference'] - data_tr['First_difference'].shift(1)
adf_check(data_tr['Seconde_difference'].dropna())
```

(3)

```
Test de Dickey_fuller Augmenté (adf)
ADF Test Statistic:-3.4866647464447778
p-value:0.008336249553514858
#lags used:11
Number of observations used:46
Forte preuve contre l'hypothèse nulle
Rejette l'hypothèse nulle
Les données n'ont pas de racine unitaire et sont donc stationnaires
```

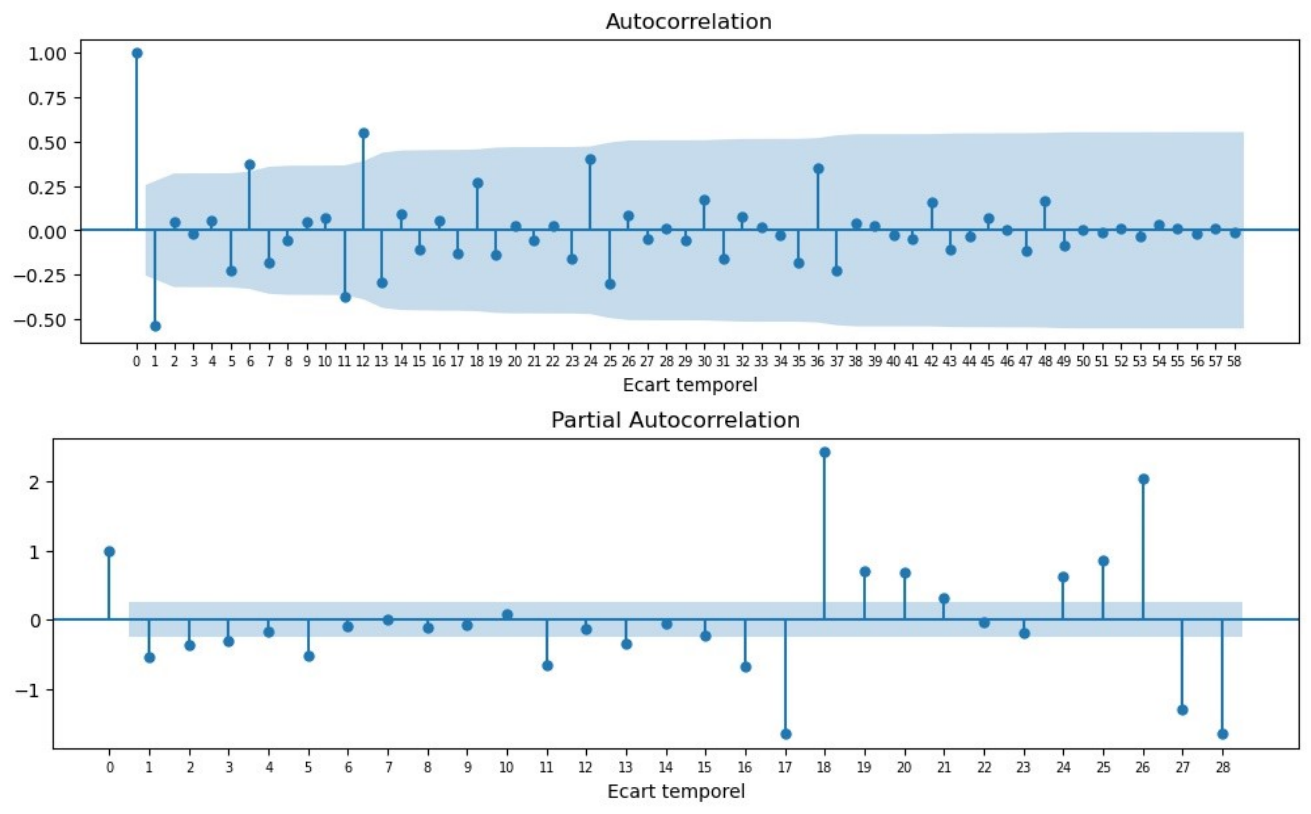
```
#data_tr['Seasonal_difference'] = data_tr['cons_nodju'] - data_tr['cons_nodju'].shift(12)
adf_check(data_tr['Seasonal_difference'].dropna())
```

(4)

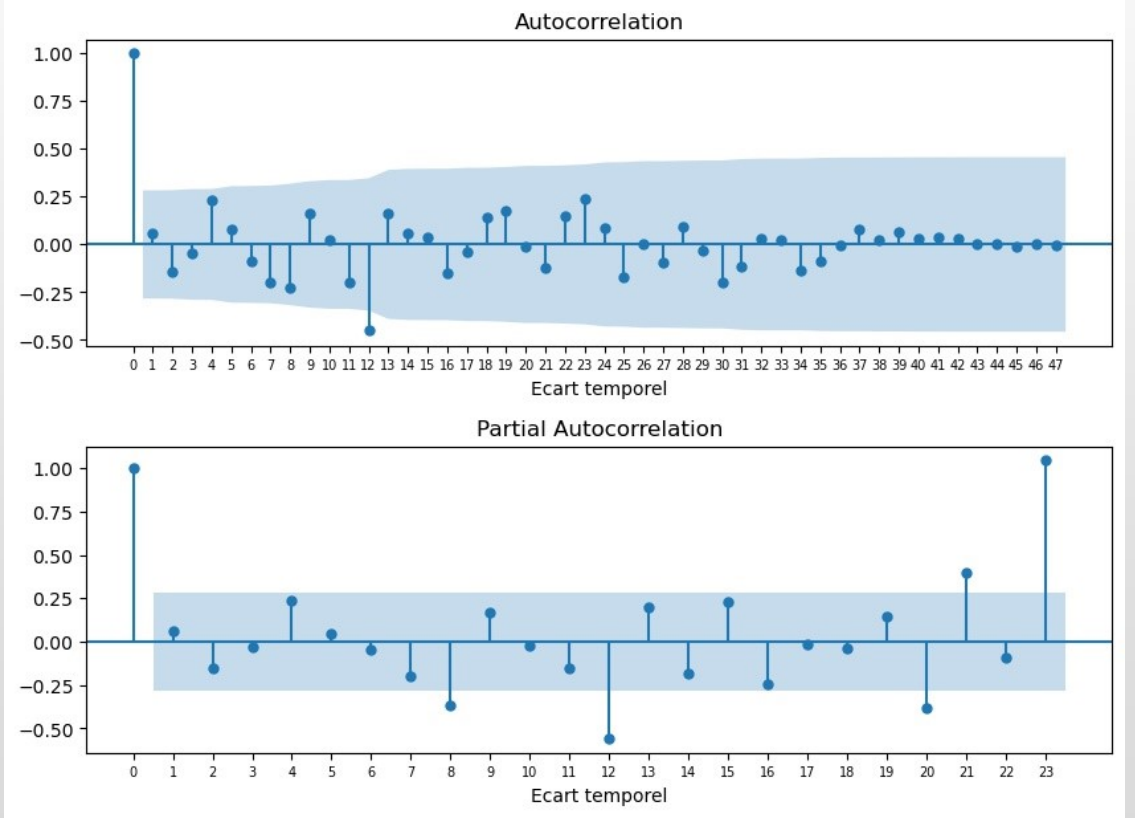
```
Test de Dickey_fuller Augmenté (adf)
ADF Test Statistic:-6.3852231099343575
p-value:2.172594723587028e-08
#lags used:0
Number of observations used:47
Forte preuve contre l'hypothèse nulle
Rejette l'hypothèse nulle
Les données n'ont pas de racine unitaire et sont donc stationnaires
```

Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode SARIMA sur la série temporelle.

Acf et Pacf pour paramétrage manuel SARIMA



1 décalage pour paramétrage manuel (p,d,q) : (3,1,6)



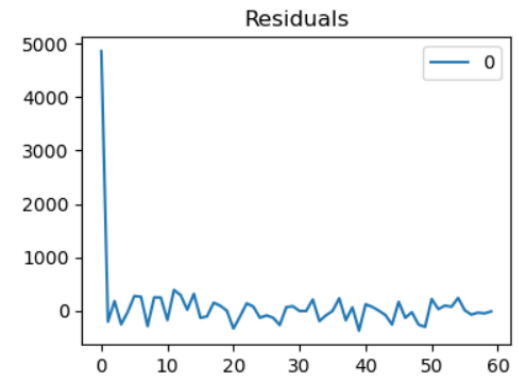
12 décalages pour paramétrage manuel (P,D,Q) : (1,0,1)

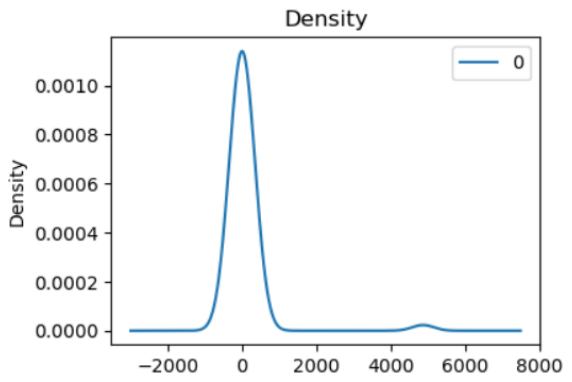
Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode SARIMA sur la série temporelle.

Création d'un model SARIMA manuel:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model1 = SARIMAX(np.asarray(data_tr['cons_nodju']), order=(3,1,6),seasonal_order = (1,0,1,12))
result1 = model1.fit()
print(result1.summary())
#sarimax_manuel
```

Test des résidus du modèle:





	0
count	60.000000
mean	77.539427
std	654.294531
min	-378.993822
25%	-136.337945
50%	-6.448484
75%	136.861642
max	4865.246987

coefficient d'asymetrie: 6.837703462443205
coefficient d'aplatissement: 50.659579138636055
p-value : 5.246683435069588e-15 ,la p-value est inférieure à un niveau alpha choisi (0.05), alors l'hypothèse nulle est rejeté
e (i.e. il est improbable d'obtenir de telles données en supposant qu'elles soient normalement distribuées).

Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode SARIMA sur la série temporelle.

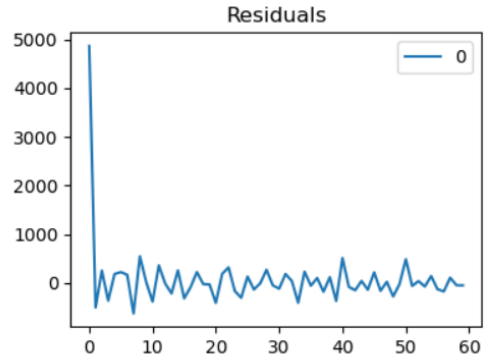
Création d'un model SARIMA automatisé:

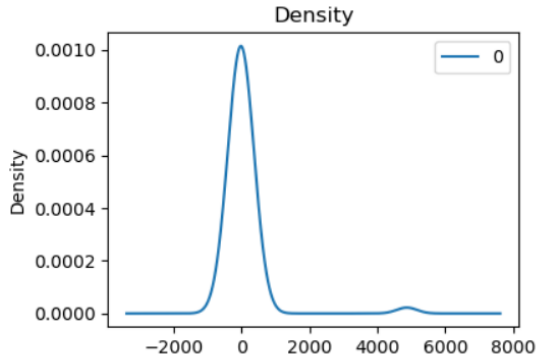
```
train,test = np.array(data_tr['cons_nodju']),np.array(data_te['cons_nodju'])

for p in p_values:
    for d in d_values:
        for q in q_values:
            der = (p,d,q)
            for P in P_values:
                for D in D_values:
                    for Q in Q_values:

                        try:

                            model = SARIMAX(train, order = der , seasonal_order=(P,D,Q,12))

                            model_fit = model.fit()

                            pred_y = model_fit.forecast(12)

                            error = np.sqrt(mean_squared_error(test,pred_y))

                            print('SARIMA%s RMSE = %.2f'% (der,error),P,D,Q)
                            if error < error_min:

                                error_min = error
                                ord = der
                                P_min = P
                                D_min = D
                                Q_min = Q


                        except:

                            print('erreur p d q :', der,P,D,Q, error)
                            continue

print('valeurs trouvées :',ord,P_min,D_min,Q_min,'RMSE:',error_min)
```

valeurs trouvées: (0, 1, 0) -- 1 0 2 12

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model2 = SARIMAX(np.asarray(data_tr['cons_nodju']), order=ord,seasonal_order = (P_min,D_min,Q_min,12))
result2 = model2.fit()
print(result2.summary())
#sarimax auto
```

Test des résidus du modèle :

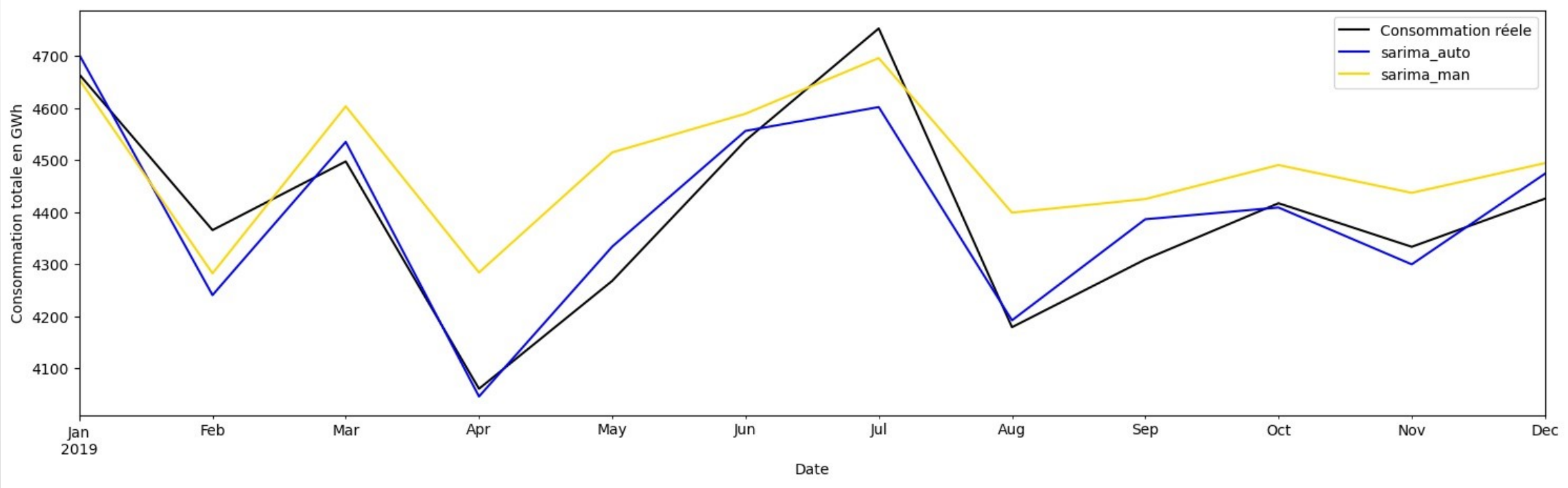




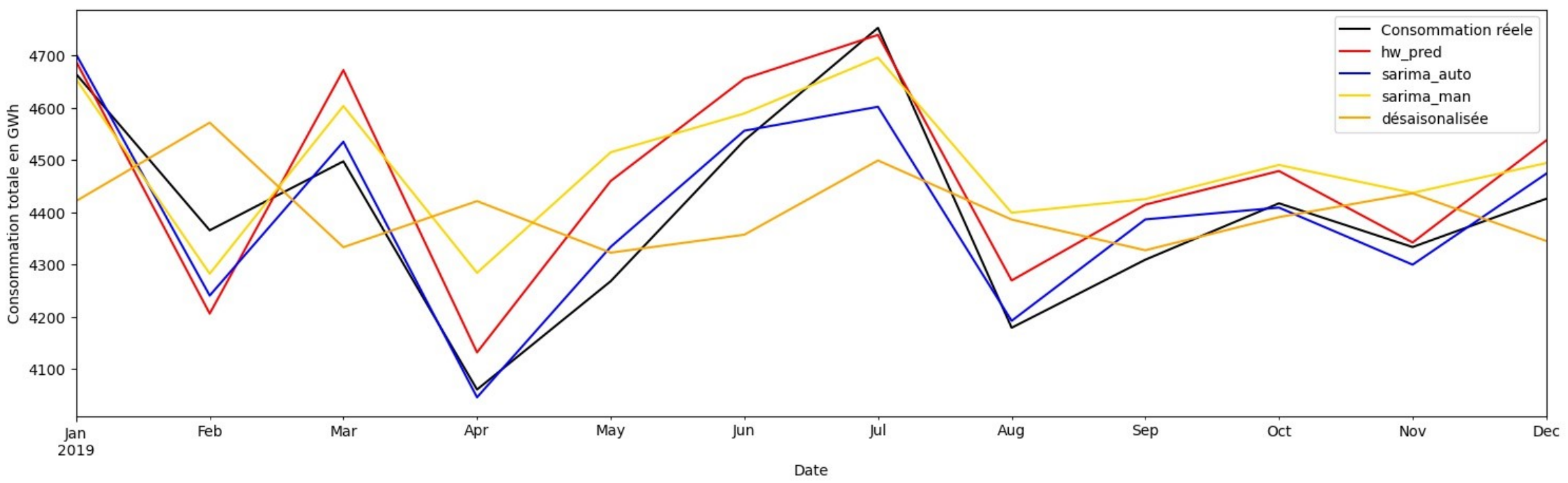

```
                0
count    60.000000
mean     67.682934
std     676.342359
min    -627.138974
25%    -153.335319
50%     -24.501995
75%     182.199634
max    4865.246987
coefficient d'asymetrie: 6.205111887914283
coefficient d'aplatissement: 44.41670215419129
p-value :  5.573201676632028e-14 ,la p-value est inférieure à un niveau alpha choisi (0.05), alors l'hypothèse nulle est rejeté
e (i.e. il est improbable d'obtenir de telles données en supposant qu'elles soient normalement distribuées).
```

Prévision de la consommation (corrigée de l'effet température) sur un an, en utilisant la méthode SARIMA sur la série temporelle.

Prévisions des modèles SARIMA et consommation réelle



Comparatif de tous les modèles




```
hw_pred rmse : 111.16591931854059
sarima_man rmse : 134.73046088733287
sarima_auto rmse : 68.22214568127349
cons_nodjusea rmse : 187.14021867750367
```

Conclusion

La méthode SARIMA a fourni les prédictions les plus proches de la réalité, sous réserve d'une optimisation des autres modèles.