

# Formation Data Analyst


Projet 9 : Prédisez la demande en électricité

23/09/2021

Aïssa MOUACHA



# SOMMAIRE

- ❑ Introduction
  - ❑ Data treatment et statistique descriptive
  - ❑ M1 : Correction des données de l'effet température
  - ❑ M2 : Désaisonnalisation des données corrigées
  - ❑ M3 : Prédiction de la consommation par Holt-Winters et SARIMA
  - ❑ Conclusion
- 

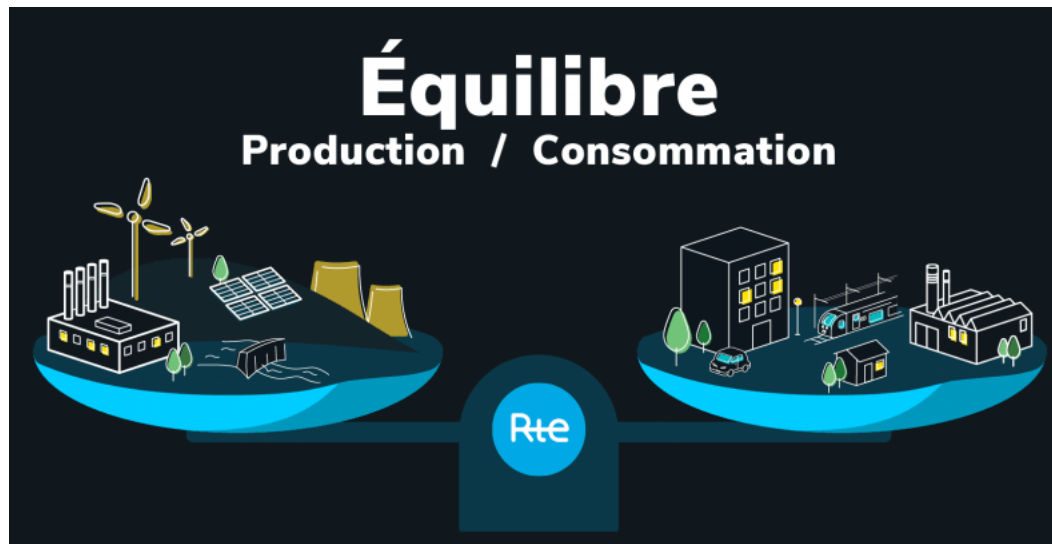
Vous êtes employé chez Enercoop, société coopérative spécialisée dans les énergies renouvelables, qui s'est développée grâce à la libéralisation du marché de l'électricité en France.



**Stratégie** : Corrigez et modélisez les données à l'aide de 2 méthodes.

**Objectif** : Prédire la consommation en électricité.

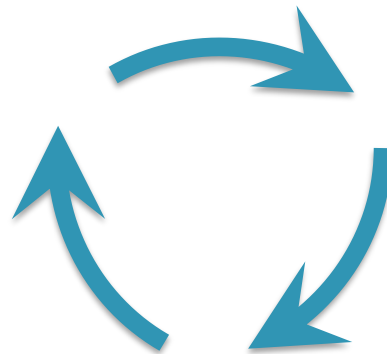
**Source** : Site CEGIBAT + RTE



## Capacités de stockage limitées

**surveiller** en permanence  
le réseau

**maîtriser** les flux



**anticiper** les évolutions de  
la consommation électrique  
à court, moyen et long  
terme.

# Statistique descriptive

*Décrire, résumer, représenter la donnée*

## Tables de l'étude

# Consommations mensuelles

Source : <https://www.rte-france.com/eco2mix/telecharger-les-indicateurs>

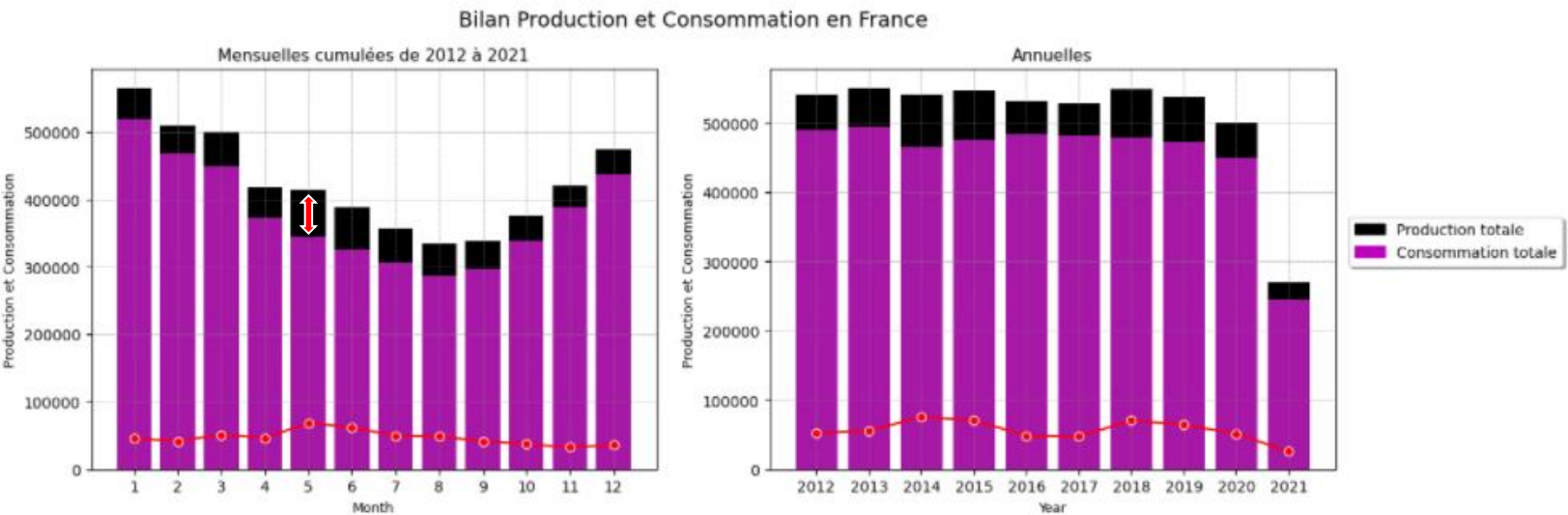
# Calcul des Degrés Jours Unifiés (DJU\*)

Source : <https://cegibat.grdf.fr/simulateur/calcul-dju>

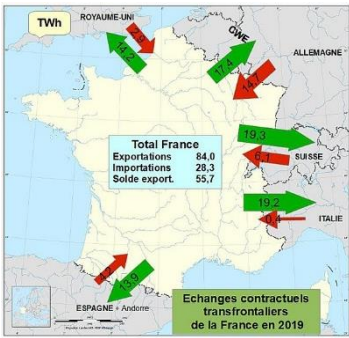
\* : valeur représentative de l'écart entre la température d'une journée donnée et un seuil de température préétabli

# Données RTE de consommations électriques

- Illustration du difficile équilibre à atteindre

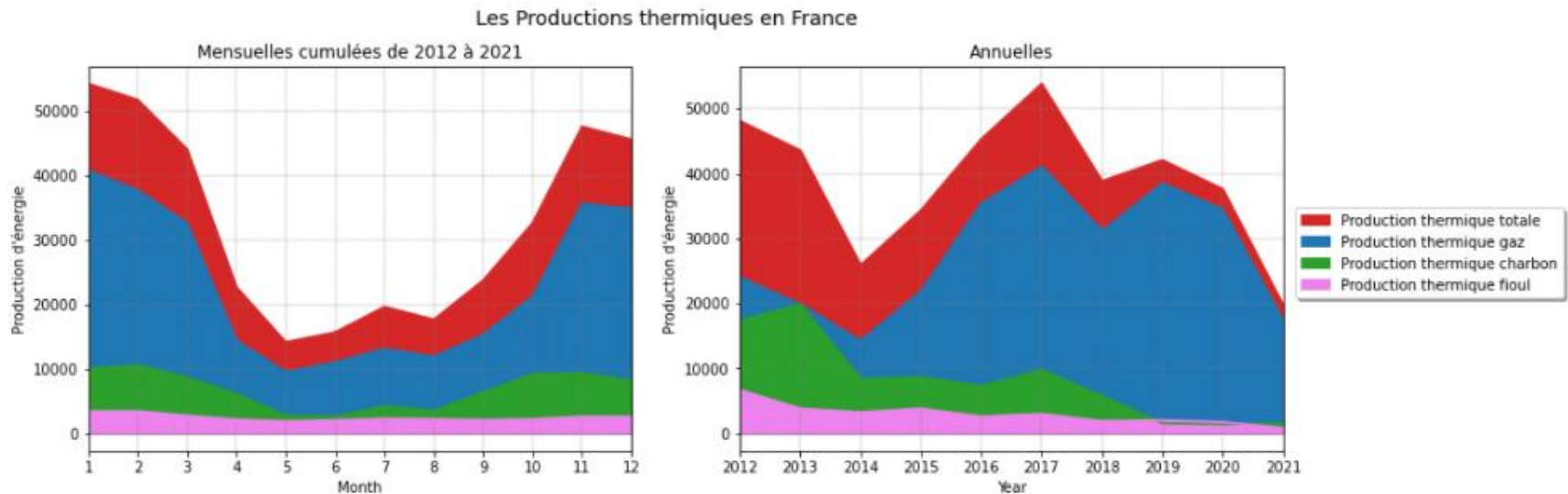


- Surproduction destinée aux échanges régionaux, transfrontaliers



## Données DJU : présentation

- Fort impact de la météo sur la consommation comme illustré ci-dessous



- Le but étant de s'affranchir des effets de chauffage liés aux « changements climatiques » pour obtenir une consommation dite corrigée.
- Utilisation de l'outil CEGIBAT



# Données DJU : téléchargement

CALCUL DES DJU

Étape 1

Étape 2

Résultats

1. Indiquez la station météo

31 - TOULOUSE-BLAGNAC

Choix d'une ville par région

CALCUL DES DJU

Étape 1

Étape 2

Résultats

2. Sélectionnez la méthode de calcul (en savoir plus sur les méthodes de calcul)

Météo

Professionnels de l'énergie

3. Sélectionnez le type d'usage

Chauffage

Climatisation

4. Sélectionnez la température de référence

16°

17°

18°

19°

20°

21°

22°

23°

24°

25°

26°

27°

28°

29°

30°

5. Période de chauffage

Date de début

01/01/09

Date de fin

15/06/21

Les DJU sont proposés sur la période du 01/01/2009 au 15/06/2021.

J'accepte les conditions d'utilisation de l'outil

Lancer la simulation

Extraction de toutes les données "chauffage" pour une Tref=18°C  
*(on a volontairement négligé ici les données pour usage de climatisation qui sont moindres)*

CALCUL DES DJU

Étape 1

Étape 2

Résultats

Résultats

Affichage en tableau

Affichage en courbes

	Jan	Fév	Mar	Avr	Mai	Jun	Jui	Aoû	Sep	Oct	Nov	Déc	Total
2021	394	200	232	174	97	1	0	0	0	0	0	0	1 098
2020	309	223	227	80	27	13	0	0	28	135	175	323	1 539
2019	407	244	218	160	96	22	0	0	6	53	253	265	1 724
2018	276	363	261	112	76	2	0	0	3	104	211	282	1 689
2017	437	239	202	143	48	6	1	2	31	64	284	362	1 818
2016	288	277	259	152	77	10	2	0	5	93	216	335	1 712
2015	375	349	228	111	43	1	1	0	24	110	188	235	1 664
2014	290	272	227	110	82	2	1	1	7	39	147	363	1 540
2013	392	353	251	182	144	28	0	0	13	55	281	347	2 046
2012	349	473	207	184	57	6	3	0	17	85	229	311	1 922
2011	395	284	231	72	17	20	2	0	6	74	158	294	1 553

Consulter le détail des hypothèses de calcul

Recevoir les résultats

Enregistrer dans mon compte

Sauvegarde des résultats

# Données DJU : méthodologie

```
print ("Les fichiers téléchargés depuis CEGIBAT sont :")
for i in files :
    print (i)

# initialisation d'un dataframe
df = pd.DataFrame()

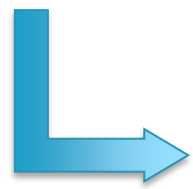
# concaténation des fichiers
for f in files:
    data = pd.read_excel(f, 'DJU - Mensuel')
    data = data.iloc[10:]
    df = df.append(data)

# set 1ère ligne comme header
df.columns = df.iloc[0]
df = df[1:]
df.rename(columns={df.columns[0]:'Year'}, inplace=True)

# conversion numérique
cols = df.columns[df.dtypes.eq('object')]
df[cols] = df[cols].apply(pd.to_numeric, errors='coerce')

# suppression des lignes intermédiaires contenant que des NAN
df = df.dropna()

# calcul de La moyenne nationale (Moy(H1x,H2x,H3)) pour La suite de L'étude
f = {'JAN': 'mean', 'FÉV': 'mean', 'MAR': 'mean', 'AVR': 'mean', 'MAI': 'mean', 'JUN': 'mean', 'JUI': 'mean', 'AOÛ': 'mean',
     'SEP': 'mean', 'OCT': 'mean', 'NOV': 'mean', 'DÉC': 'mean', 'Total': 'mean'}
df = df.groupby(['Year']).agg(f)
```



Les fichiers téléchargés depuis CEGIBAT sont :

- P9\_calcul\_DJU\_03\_09\_2021\_1.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_2.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_3.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_4.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_5.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_6.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_7.xlsx
- P9\_calcul\_DJU\_03\_09\_2021\_8.xlsx



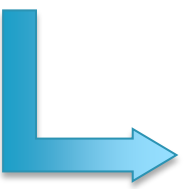
	10	JAN	FÉV	MAR	AVR	MAI	JUN	JUI	AOÛ	SEP	OCT	NOV	DÉC	Total
Year														
2009.0	456.2125	348.9875	287.2875	162.9000	65.8875	24.2875	5.9875	4.1375	27.4375	139.5625	210.4750	385.2750	2118.1750	
2010.0	478.3500	359.0250	307.0125	172.0625	128.2125	23.6625	1.0500	13.8750	58.3750	165.0000	286.1375	464.2250	2456.7250	
2011.0	395.0875	302.1125	258.8000	103.1625	51.3750	30.2250	17.6500	11.7375	23.0000	116.3000	205.0250	309.8000	1823.9875	
2012.0	350.7000	455.8125	224.9750	210.6500	83.1500	26.9250	13.9500	3.4875	48.2375	127.8125	258.9750	336.6375	2141.0375	
2013.0	406.8000	388.7000	327.0250	206.9875	153.1250	40.7250	1.3875	4.7750	33.9625	85.8750	289.6375	354.8625	2293.6875	

# Données DJU : traitement

```
# pivot
stacked = df.stack()
stacked = pd.DataFrame(data=stacked)

# superposition
stacked.rename(columns={stacked.columns[0]: 'DJU_effet_T'}, inplace=True)
df = stacked.reset_index()
df

# renaming & conversion
df.rename(columns={10: 'Month'}, inplace=True)
df
di = {'JAN':1, 'FÉV':2, 'MAR':3, 'AVR':4, 'MAI':5, 'JUN':6, 'JUI':7, 'AOÛ':8, 'SEP':9, 'OCT':10, 'NOV':11, 'DÉC':12,}
df['Month'].replace(di, inplace=True)
df['Year'] = df['Year'].astype('int64')
```



	Year	Month	DJU_effet_T
0	2009	1	456.2125
1	2009	2	348.9875
2	2009	3	287.2875
3	2009	4	162.9000
4	2009	5	65.8875
...	...	...	...

	Year	Month	Consommation totale
0	2012	1	51086
1	2012	2	54476
2	2012	3	43156
3	2012	4	40176
4	2012	5	35257
...	...	...	...



Obtention table compatible avec la table de la consommation électrique



```
Sample_final = pd.merge(table, df_DJU, how="left", on=['Year', 'Month'])
```

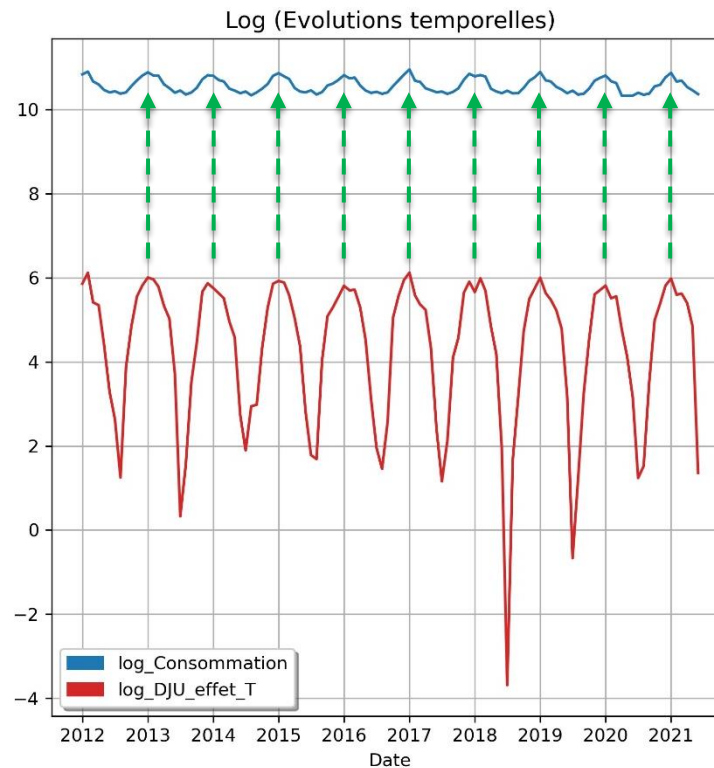
- A ce stade, la consommation intègre l'effet de la température (chauffage)
  - Objectif : retirer cet effet sur nos données

## Correction des données

*Mise en évidence de l'effet de température*

## Influence des DJU sur la consommation

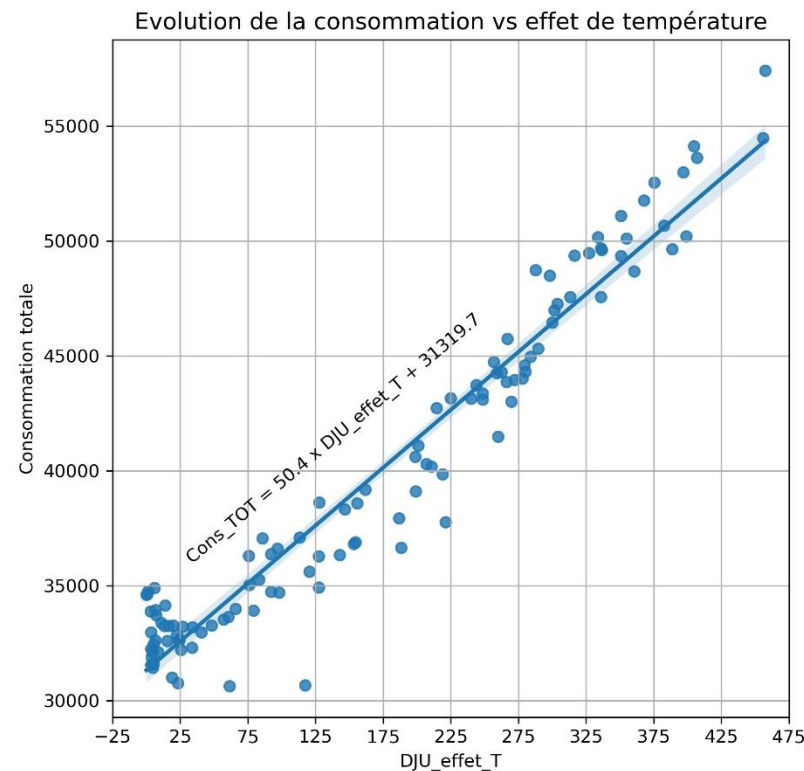
- Le passage en log des timeseries permet de mettre en évidence le phasage entre les variables **DJU** et **consommation** → lien évident



## Mise en évidence de la corrélation

- Le tracé de la consommation en fonction des DJU confirme la corrélation qui existe et donc le réel impact des DJU sur la consommation.
- Un regplot via Seaborn permet de caractériser cette corrélation

```
corr = sns.regplot(x="DJU_effet_T", y="Consommation totale", data=Sample_final, color="tab:blue")  
slope, intercept, r_value, p_value, std_err = stats.linregress(x=corr.get_lines()[0].get_xdata(),  
y=corr.get_lines()[0].get_ydata())
```



# Régression linéaire

```
Y = Sample_final["Consommation totale"]
DJU_effet_T = Sample_final["DJU_effet_T"]

# régression
model = ols('Y ~ DJU_effet_T', data=Sample_final).fit()
```

*méthode des moindres carrés ordinaire*



OLS Regression Results						
=====						
Dep. Variable:	Y	R-squared:	0.941			
Model:	OLS	Adj. R-squared:	0.941			
Method:	Least Squares	F-statistic:	1793.			
Date:	Thu, 23 Sep 2021	Prob (F-statistic):	9.25e-71			
Time:	09:56:54	Log-Likelihood:	-1010.4			
No. Observations:	114	AIC:	2025.			
Df Residuals:	112	BIC:	2030.			
Df Model:	1					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
Intercept	3.132e+04	257.105	121.817	0.000	3.08e+04	3.18e+04
DJU_effet_T	50.3519	1.189	42.349	0.000	47.996	52.708

Coefficient-poids

Qualité du fit 94% modèle performant

DJU statistiquement significative

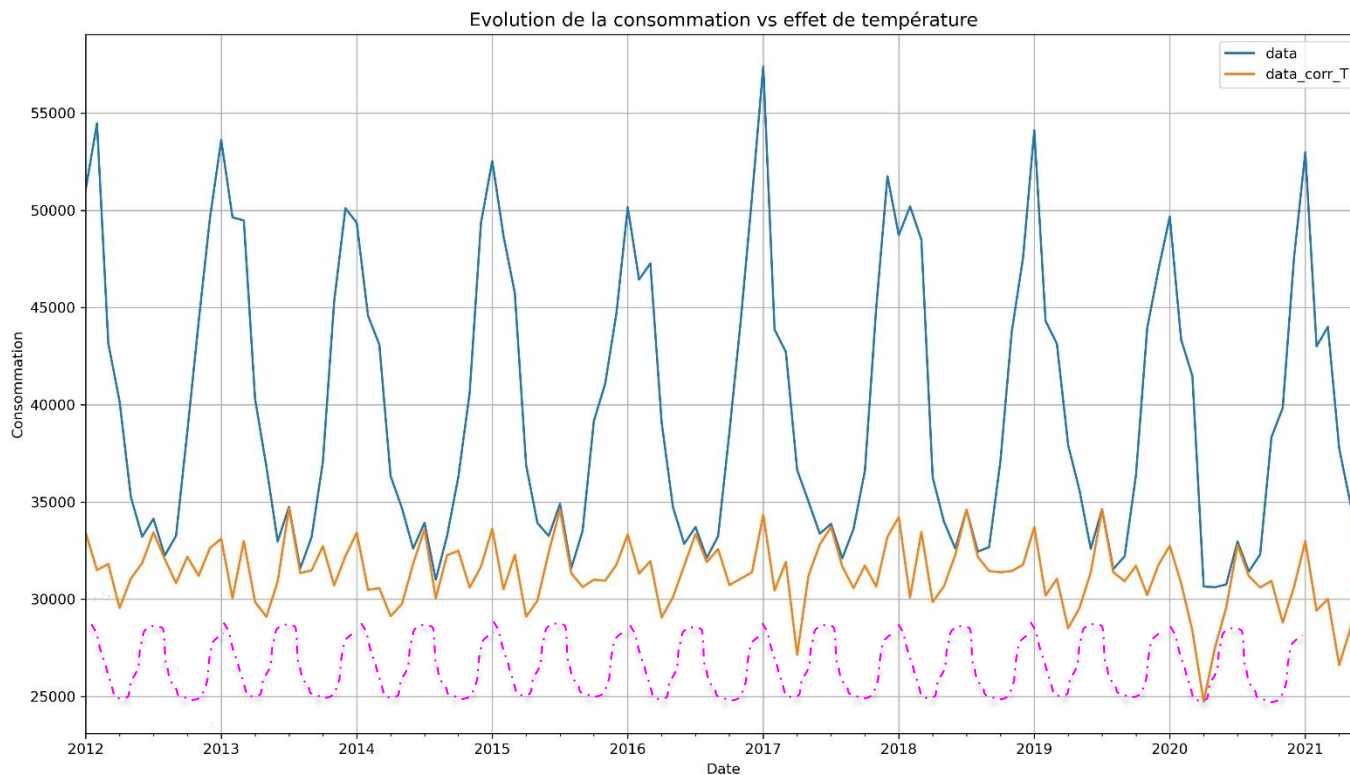
- La régression linéaire vient confirmer la relation établie précédemment
- On peut donc écrire :  $Y = aX + b + \varepsilon$  sous cette forme  $CONS = 50 DJU + 31320 + \varepsilon$

## Correction des données

- L'étape de correction consiste à soustraire à la variable **consommation** la contribution ou l'effet **DJU** pondéré par le coefficient obtenu (c), soit :

coef\_DJU\_T c = 50.4

```
# on corrige la consommation de l'effet de température (à l'aide du coef 'pondération')  
Sample_final["Consommation totale corrigée"] = Sample_final["Consommation totale"] - c * Sample_final["DJU_effet_T"]
```





- A ce stade, obtention de la timeserie consommation corrigée
- Objectif : décomposer cette série afin d'en extraire la tendance, l'effet saisonnier et une contribution résiduelle

## Désaisonnalisation des données

*Décomposer une timeserie en 3 contributions*

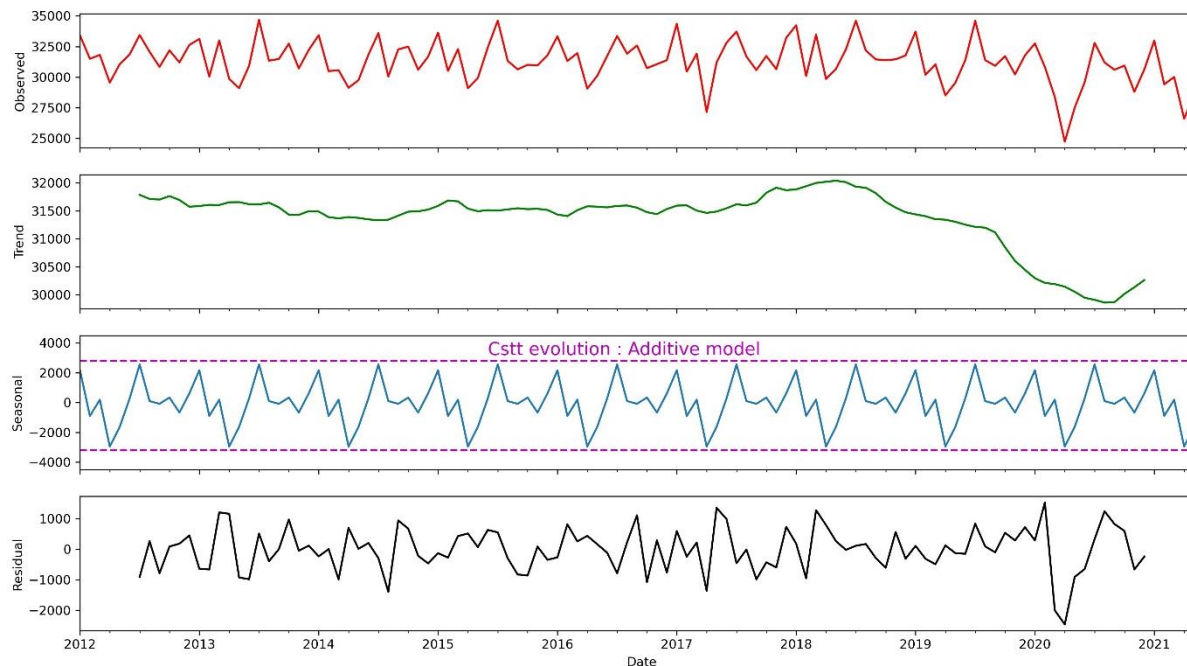
$$* X_t = T_t + S_t + \varepsilon_t$$

\* : pour la suite de notre projet on considèrera le modèle de décomposition comme additif (voir ci-après)

# Décomposition sous Statsmodels

```
import statsmodels.api as sm

X = g['data_corr_T']
model = sm.tsa.seasonal_decompose(X,
                                  model='additive', #'multiplicative'
                                  filt=None,
                                  period=12,
                                  two_sided=True,
                                  extrapolate_trend=0,
                                  #extrapolate_trend='freq',
                                  )
```



Série temporelle corrigée initiale

Tendance à la baisse sur les 3 dernières années

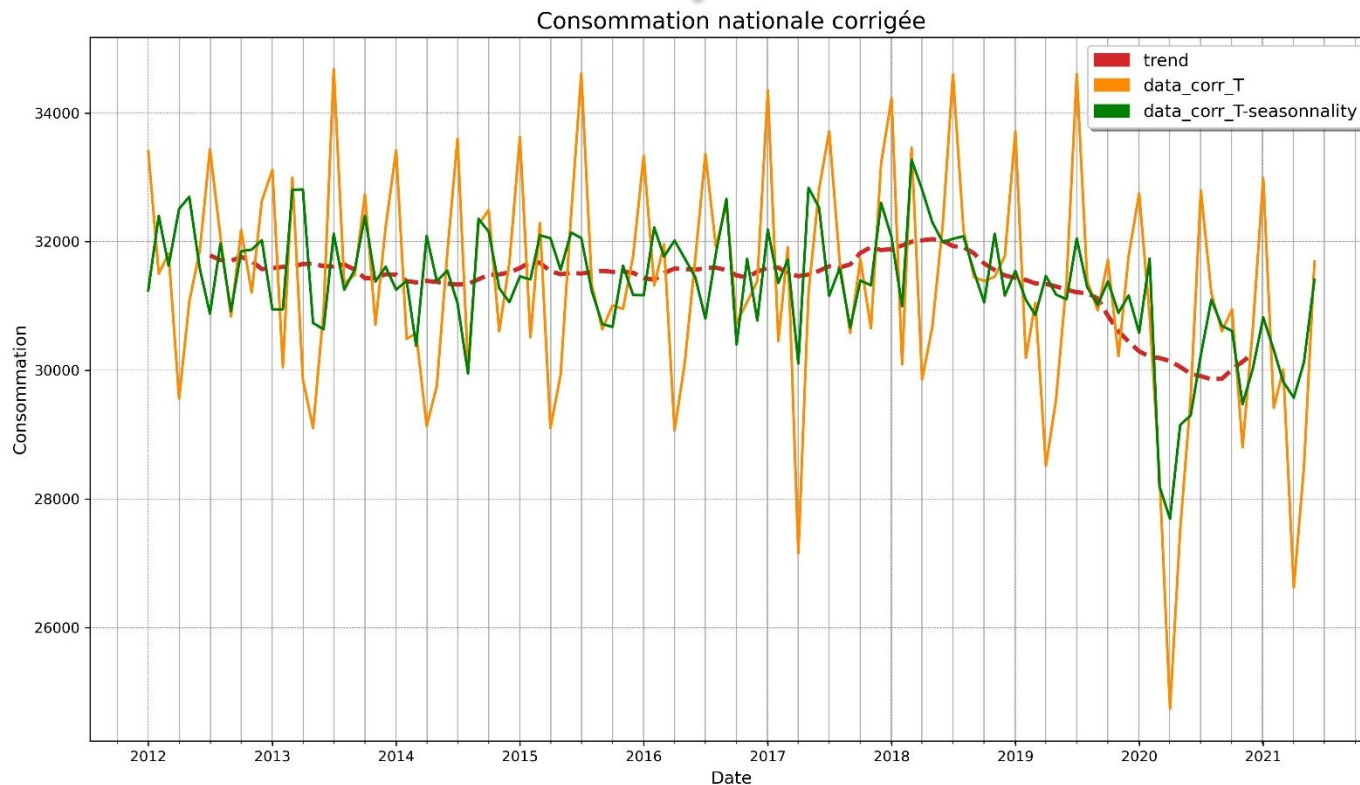
Effet saisonnier régulier et constant

Bruit (variations aléatoires dans la série)

# Désaisonnalisation des données

- L'étape de désaisonnalisation consiste à soustraire à la variable **consommation** corrigée la contribution de l'effet saisonnier mise en évidence soit :

```
# on corrige la consommation corrigée de l'effet saisonnier grâce à la décomposition  
decompose["data_corr_T-seasonnality"] = decompose["data_corr_T"].values - decompose["seasonal"].values
```



- A ce stade, obtention de la timeserie consommation corrigée et désaisonnalisée (il en résulte une série lissée, épurée où les maxima/minima sont atténués)
  - Objectif : Prédire la consommation en électricité à l'aide de 2 méthodes

## Prédiction de la consommation

*Lissage exponentiel Holt–Winters*

*Modèle SARIMA*



*Une prévision  
n'est qu'une  
fonction du  
passé de la  
série*

# Méthodes de lissage exponentiel

- Parmi les méthodes d'approximation on retrouve :
  - Lissage exponentiel simple (LES) : timeserie approximable autour de T (date) par une constante

a

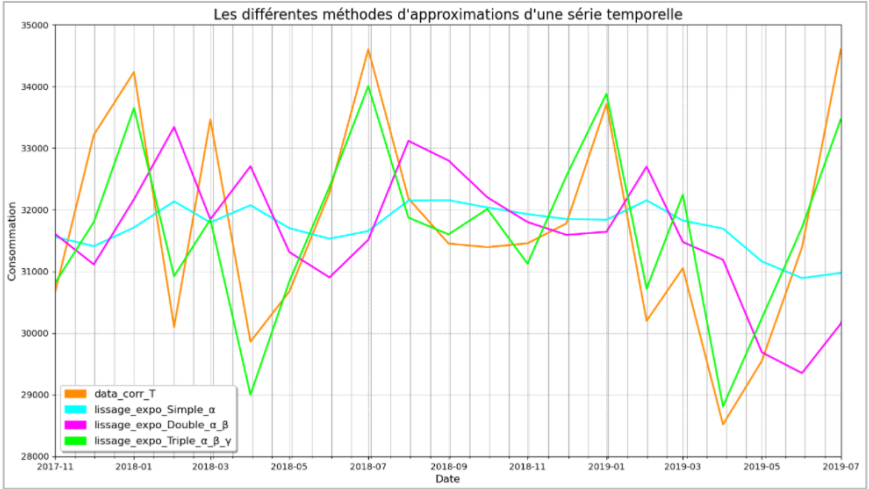
- Lissage exponentiel double (LED) : timeserie approximable autour de T par une droite

$aT + (t-T)bT$



- Lissage exponentiel triple (H-W) : timeserie approximable autour de T par une droite

$aT + (t-T)bT + ST$



*α, b et S étant communément appelés α, β, γ et représentent les paramètres de lissage de la méthode H-W.  
Ces paramètres sont choisis pour minimiser la somme au carré des erreurs de prédiction*

## M thodes de lissage exponentiel

- A l'aide de la librairie Statsmodels, on peut faire appel   des fonctions qui nous permettent de g n rer ces approximations et ainsi voir quels sont les meilleurs coefficients retenus par la m thode

```
# Import M thodes de lissage
from statsmodels.tsa.holtwinters import SimpleExpSmoothing # single exponential smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing # double and triple exponential smoothing

# Obtention des Coefficients ( ,  ,  ) issus de l'algo avec "optimization=True"
model_simple = SimpleExpSmoothing(decompose['data_corr_T']).fit()
model_double = ExponentialSmoothing(decompose['data_corr_T'], trend='add').fit()
model_triple = ExponentialSmoothing(decompose['data_corr_T'], trend='add', seasonal='add', seasonal_periods=12).fit()
# model.summary()
res_df = pd.DataFrame([model_simple.params, model_double.params, model_triple.params])
```



	$\alpha$	$\beta$	$\gamma$
	smoothing_level	smoothing_trend	smoothing_seasonal
0	0.167691	NaN	NaN
1	0.463084	0.139690	NaN
2	0.148838	0.000529	0.228686

Valeurs proches de 0   poids important des anciennes valeurs

Valeurs proches de 1   poids important des valeurs r centes

- L'utilisateur peut choisir d'imposer ces coefficients (par d faut optimis  par Statsmodels)

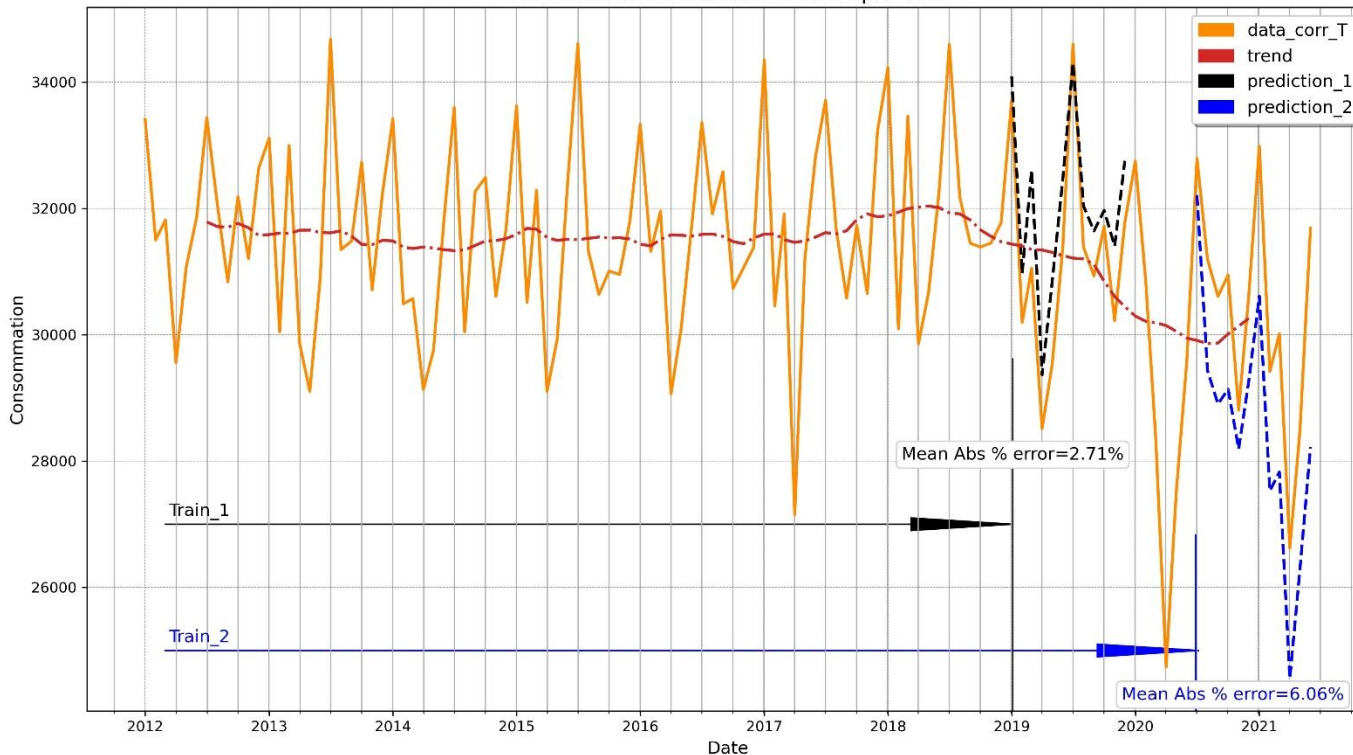
# Méthode de Holt-Winters : modélisation

- 2 modèles créés à partir de 2 sets d'apprentissage (train\_i)

```
mod_1 = ExponentialSmoothing(train_decompose_1['data_corr_T'],trend='add',seasonal='add',seasonal_periods=12).fit()  
prediction_1 = mod_1.forecast(len(test_decompose_1))  
mod_2 = ExponentialSmoothing(train_decompose_2['data_corr_T'],trend='add',seasonal='add',seasonal_periods=12).fit()  
prediction_2 = mod_2.forecast(len(test_decompose_2))
```

- Prédiction sur l'année consécutive (test\_i) et comparaison aux données réelles

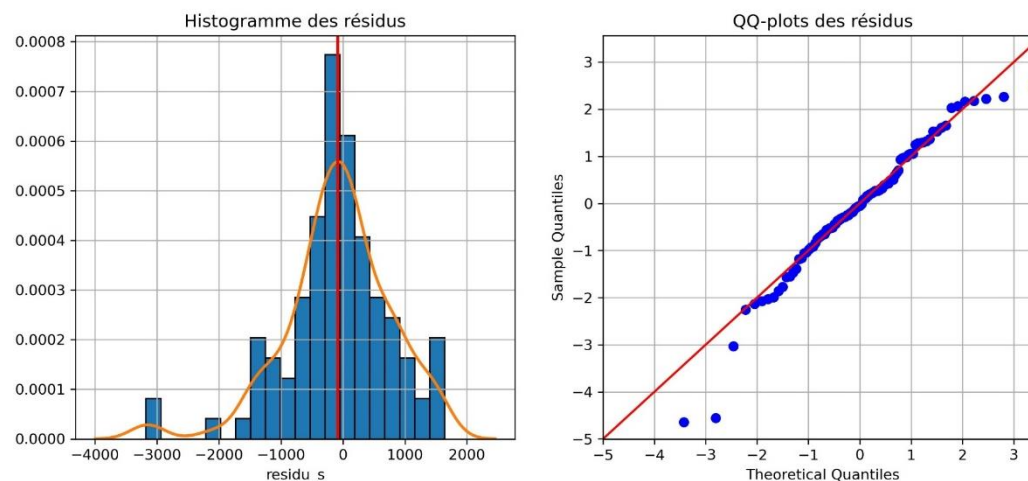
Prévision de la consommation par HW



- Prédictions fidèles aux observations réelles
- Impact outliers sur la précision du modèle
- Performance modèle relativement bonne (à comparer avec SARIMA)

# Méthode de Holt-Winters : analyse des résidus

- La forme de la distribution des résidus est à 96% similaire à une loi normale selon le test de Shapiro-Wilk → normalité



- Quel que soit le modèle retenu, un test de Ljung-Box vient confirmer qu'on est en présence d'un bruit blanc ( $pvalue > 5\%$ )

```
print(sm.stats.acorr_ljungbox(model_triple_pred_2.resid, lags=[1,2,3,4,5,6,7,8,9,10,11,12], return_df=True))
print('\n\b[6;31;40m', "On ne rejète pas l'hypothèse H0 d'un bruit blanc avec une pvalue > 5%.", '\n\b[0m')
# reject of H0 => residuals are dependent.
```

	lb_stat	lb_pvalue
1	0.748692	0.386891
2	1.128541	0.568775
3	1.290406	0.731412
4	2.053428	0.725933
5	2.053439	0.841702
6	2.243252	0.896023
7	2.520884	0.925516
8	5.436457	0.710069
9	5.600604	0.779130
10	5.904840	0.823194
11	5.976590	0.874922
12	6.764818	0.872756

On ne rejète pas l'hypothèse H0 d'un bruit blanc avec une pvalue > 5%.



## Modèle SARIMA

- On appelle processus SARIMA (Seasonal AutoRegressive Integrated Moving Average) d'ordre  $(p,d,q)(P,D,Q)_s$  , un processus qui permet de modéliser des séries qui présentent une saisonnalité.
- Les étapes qui vont suivre :
  - Stationnarisation de la série (rendre la série modélisable, ici par différenciation)
  - Détermination des paramètres optimaux pour modèle SARIMA
  - Modélisation avec SARIMAX
  - Analyse du bruit (résidus)
  - Visualisation des 2 méthodes de prédiction (année  $n-1$ ) et metrics associées
  - Prédiction à posteriori (année  $n+1$ )

# Modèle SARIMA : Stationnarisation par différenciation

- Vérification stationnarité par test de Dickey-Fuller (autres tests existant KPSS, Phillips-Perron)

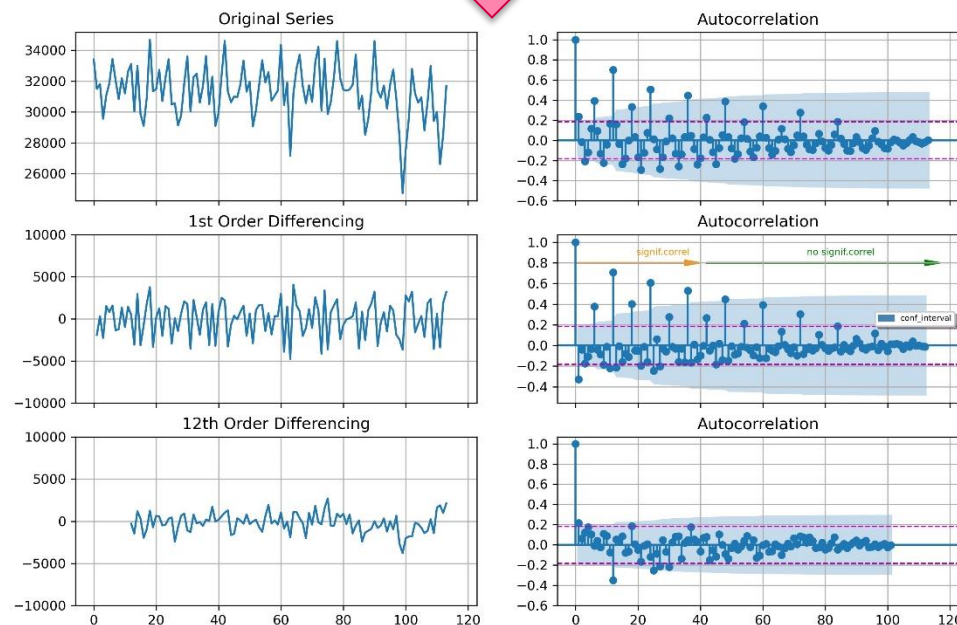
```
#ADF-test(Original-time-series)
result = adfuller(data.dropna())
print('ADF Statistic Original-time-series: %f' % result[0])
print('p-value: %f' % result[1], '\x1b[6;31;40m', "si > 5% => instationnarité => différenciation nécessaire", '\x1b[0m \n')

ADF Statistic Original-time-series: -0.597431
p-value: 0.871561 \x1b[6;31;40m si > 5% => instationnarité => différenciation nécessaire
```



```
#ADF-test(differenced-time-series)
result = adfuller(data.diff().dropna())
print('ADF Statistic differenced-time-series: %f' % result[0])
print('p-value: %f' % result[1], '\x1b[6;32;40m', " < 5% => stationnarité établie => d = 1", '\x1b[0m')

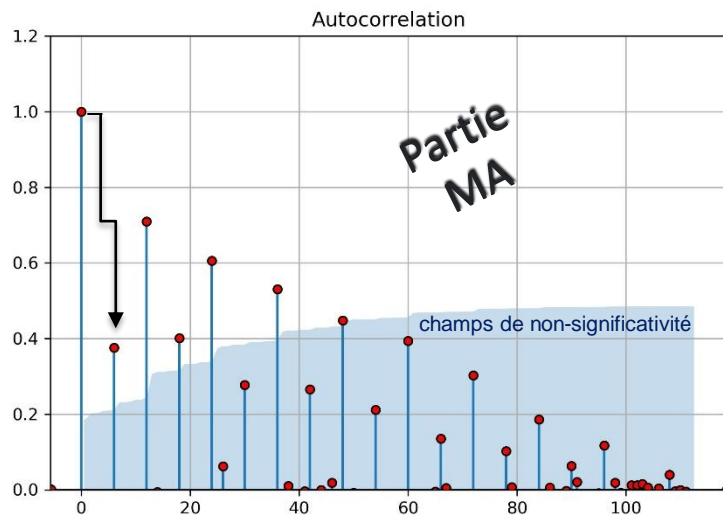
ADF Statistic differenced-time-series: -6.198265
p-value: 0.000000 \x1b[6;32;40m < 5% => stationnarité établie => d = 1
```



➤  $s = 12$

# Modèle SARIMA : Autocorrélogrammes

- On fixe les paramètres du modèle ( $p, q$ ) → lecture sur autocorrélogrammes (ACF & PACF)

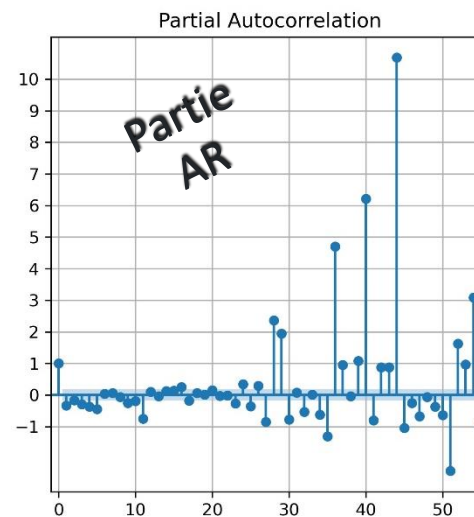


pic au lag1 → forte corrélation entre chaque valeur de série et la valeur précédente,  
pic au lag2 → forte corrélation entre chaque valeur et la valeur apparaissant deux points auparavant

...



$q$



Au décalage  $k$ , il s'agit de la corrélation entre les valeurs de séries séparées par  $k$  intervalles, compte tenu des valeurs des intervalles intermédiaires



$p$

# Modèle SARIMA : Détermination des paramètres optimaux

- Utilisation d'un programme qui va balayer différentes combinaisons paramétriques et va retourner ces dernières triées par metrics \*AIC/BIC croissantes.

```
# Import Packages #
import itertools
import statsmodels.api as sm

# Define the p, d and q parameters to take any value between 0 and 3 (exclusive)
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (12 in the 's' position as we have monthly data)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

# Run Grid Search with pdq and seasonal pdq parameters and get the best BIC value #
def sarimax_gridsearch(ts, pdq, pdqs, maxiter=50, freq='M'):
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(ts, # this is your time series you will input
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)

                output = mod.fit(maxiter=maxiter)
                ans.append([comb, combs, output.aic, output.bic])
                print('SARIMAX {} x {}12 : AIC Calculated ={}, BIC Calculated ={}'.format(comb, combs, output.aic, output.bic))
            except:
                continue

# Convert into dataframe
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic', 'bic'])

# Sort and return top 10 combinations
ans_df = ans_df.sort_values(by=['bic'],ascending=True)[0:10]
```



	pdq	pdqs	aic	bic
27	(0, 1, 1)	(0, 1, 1, 12)	1248.858956	1255.811421
31	(0, 1, 1)	(1, 1, 1, 12)	1250.802645	1260.072598
59	(1, 1, 1)	(0, 1, 1, 12)	1250.814642	1260.084595
63	(1, 1, 1)	(1, 1, 1, 12)	1252.726683	1264.314123
43	(1, 0, 1)	(0, 1, 1, 12)	1274.401134	1283.724068
11	(0, 0, 1)	(0, 1, 1, 12)	1283.013574	1290.005774
15	(0, 0, 1)	(1, 1, 1, 12)	1282.521567	1291.844500
47	(1, 0, 1)	(1, 1, 1, 12)	1281.565171	1293.218838
51	(1, 1, 0)	(0, 1, 1, 12)	1286.947159	1293.939359
55	(1, 1, 0)	(1, 1, 1, 12)	1288.193100	1297.516033

\* Le critère AIC pénalise les modèles comportant trop de variables, et évite le sur-apprentissage  
 Le critère BIC pénalise les modèles comportant trop de variables, et évite le sur-paramétrage

# Modèle SARIMA : Détermination optimale des paramètres

- Après étude de différents tests parmi les combinaisons obtenues  
 ➔ sélection du modèle le plus pertinent

```
# print('version statmodel',sm.__version__)
from statsmodels.tsa.statespace.sarimax import SARIMAX
my_order = (0,1,1) # p, d, q
my_seasonal_order = (0,1,1,12) # P, D, Q, s # Good MODEL with Train_2 in gridsearch
# run fit on same train sample as H-W fit
model = SARIMAX(train_decompose_2['data_corr_T'], order=my_order, seasonal_order=my_seasonal_order,
                 enforce_stationarity=False, enforce_invertibility=False).fit()
model.summary()
```



Dep. Variable:	data_corr_T		No. Observations:	102		
Model:	SARIMAX(0, 1, 1)x(0, 1, 1, 12)		Log Likelihood	-621.429		
Date:	Thu, 30 Sep 2021		AIC	1248.859		
Time:	17:28:02		BIC	1255.811		
Sample:	0		HQIC	1251.635		
				- 102		
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ma.L1	-0.7553	0.074	-10.165	0.000	-0.901	-0.610
ma.S.L12	-0.5602	0.111	-5.026	0.000	-0.779	-0.342
sigma2	8.759e+05	1.09e+05	8.067	0.000	6.63e+05	1.09e+06
Ljung-Box (L1) (Q): 0.00 Jarque-Bera (JB): 9.44						
Prob(Q): 0.99 Prob(JB): 0.01						
Heteroskedasticity (H): 2.02 Skew: -0.63						
Prob(H) (two-sided): 0.09 Kurtosis: 4.19						

Indicateurs de performance du modèle

Variables statistiquement significatives

Normalité des résidus (par test de Shapiro-Wilk)

Bruit blanc  
Pas de variance

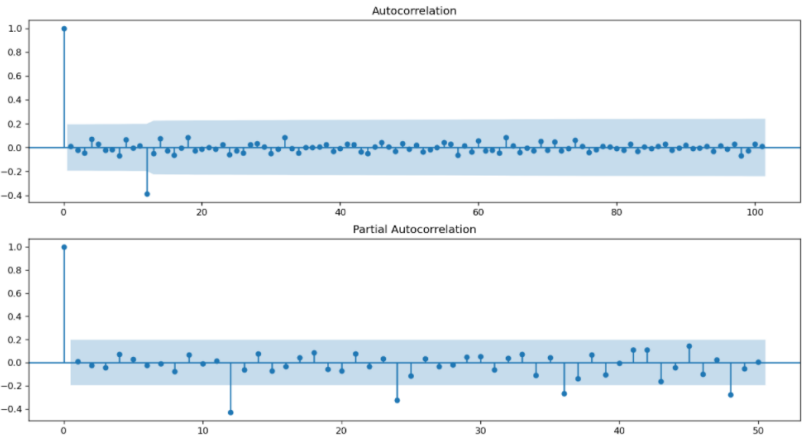
# Modèle SARIMA : Analyse du bruit (résidus)

- Normalité et bruit blanc des résidus mis en évidence par :

autocorrélogrammes  
(absence de pics significatifs)

distribution gaussienne, QQ-plots

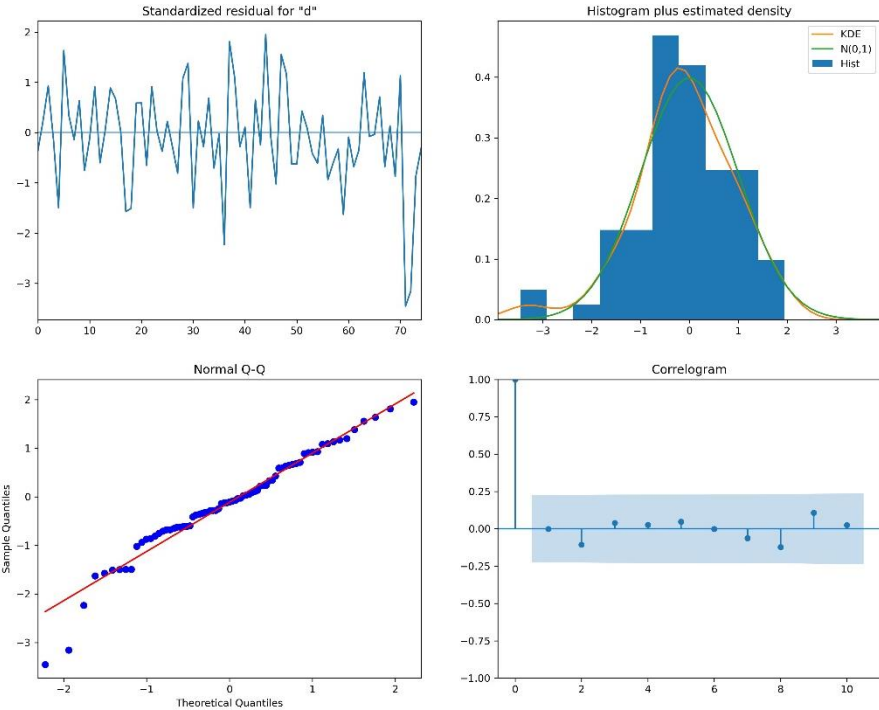
Test de Shapiro-Wilk  
(pvalue ~ 9.2<sup>-19</sup>)



```
# print(sm.stats.acorr_ljungbox(model.resid, return_df=True))
print(sm.stats.acorr_ljungbox(model.resid, lags=[1,2,3,4,5,6,7,8,9,10,11,12], return_df=True))
print('\x1b[6;31;40m', "On ne rejète pas l'hypothèse H0 d'un bruit blanc avec une pvalue > 5%.", '\x1b[0m')
# reject of H0 => residuals are dependent.
```

	lb_stat	lb_pvalue
1	0.033632	0.854492
2	0.046099	0.976774
3	0.212788	0.975499
4	0.620915	0.960711
5	0.801088	0.976964
6	0.816262	0.991631
7	0.914769	0.996089
8	1.372313	0.994631
9	1.650043	0.995873
10	1.662291	0.998335
11	1.747972	0.999204
12	18.137045	0.111592

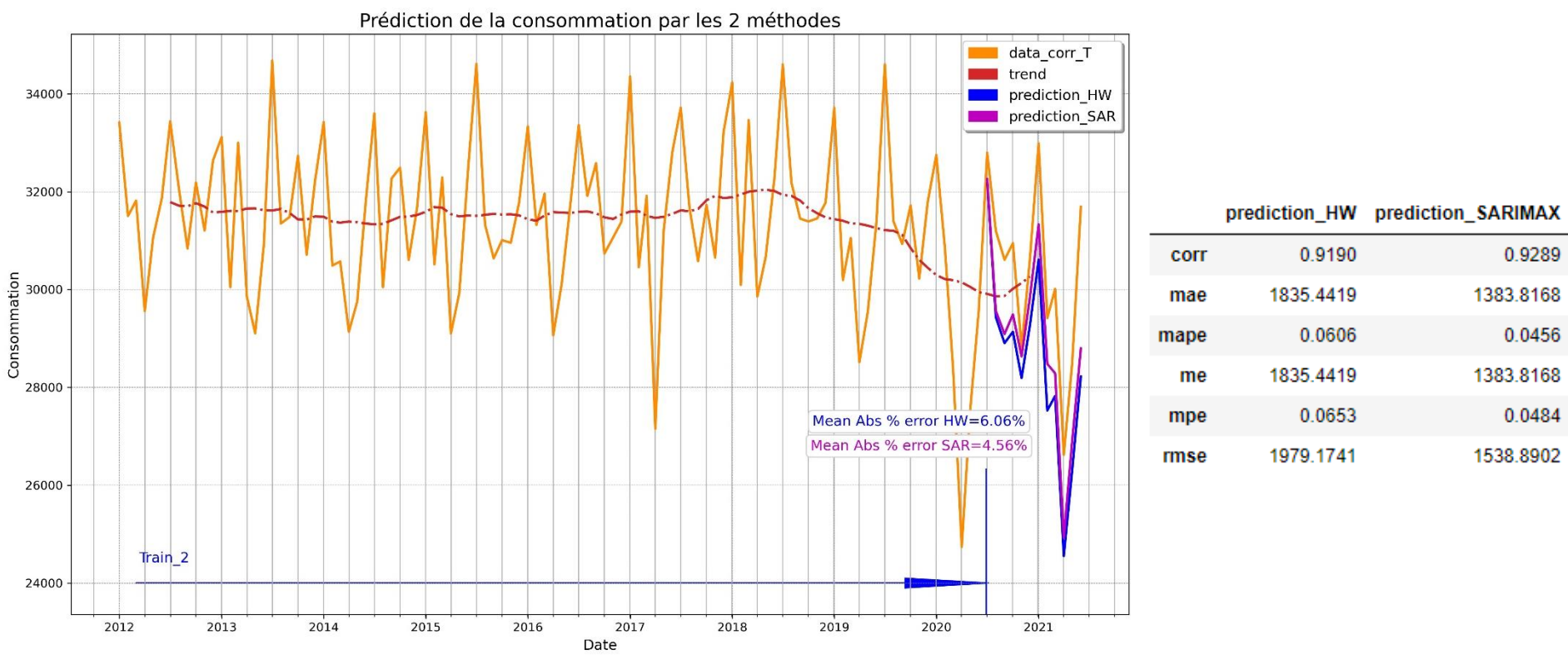
On ne rejète pas l'hypothèse H0 d'un bruit blanc avec une pvalue > 5%.



model.plot\_diagnostics(figsize=(15, 12))

# Comparaison des 2 méthodes de prédiction

- Les 2 modèles présentent des metrics « erreurs » très proches, tous deux suivent globalement l'évolution des données réelles.

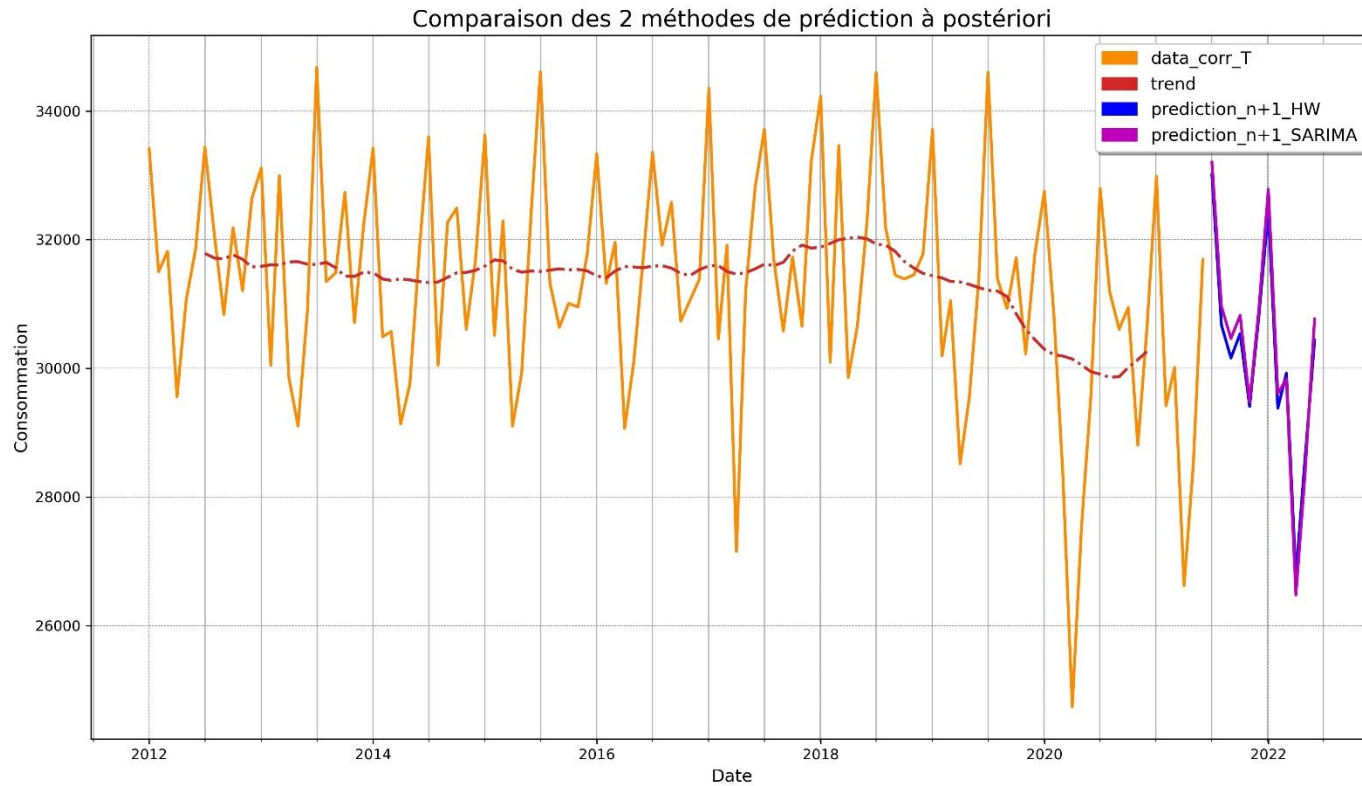


- Avantage au modèle SARIMA qui présente des metrics «  $\Delta$ pred/real » plus faibles.



## Prédiction à postériori

- Prédiction année (n+1) à partir des 2 modélisations établies



- Comme attendu, prédictions très proches



❖ Test de 2 méthodes de prédiction :

- **Résultats fidèles** aux données de consommation réelles
- **Performance** très **proche** et **satisfaisante** pour toute prédiction à postériori

❖ Si l'on devait choisir, on privilégiera le modèle SARIMA

Merci

