Projet 9_Prédiction de la demande en électricité

Parcours Data Analyst

Xuefei ZHANG_Avril 2022

Sommaire

1. Constitution de dataframes
2. Correction de l'effet température_rég linéaire
3. Désaisonnalisation: CVS
   a. régression linéaire
   b. moyennes mobiles
4. Prédiction de la consommation
   a. lissage exponentiel Holt-Winters
   b. SARIMAX
5. Analyse à posteriori

1. Constitution de dataframes

- Récupération des jeux de données appropriés
- Nettoyage de jeux de données
- Traitement de dataframes

1.1 Sources de données

1. **données mensuelles de consommation totale d'électricité**
   https://www.rte-france.com/eco2mix/telecharger-les-indicateurs
   métropole: Ile-de-France

2. **mesures de l'effet météo sur la consommation électrique: DJUchauffage, DJUclimatisation**
   https://cegibat.grdf.fr/simulateur/calcul-dju
   station météo: Paris
   seuil: 18°C
   * Le **degré jour** est une valeur représentative de l'écart entre la température d'une journée donnée et un seuil de température préétabli (18 °C dans le cas des DJU ou **Degré Jour Unifié**). ils permettent de calculer les besoins de chauffage et de climatisation d'un bâtiment.

1.2 import & traitement des jeux de données

```
---  ------       --------------  -----
 0   DJUchauffage  90 non-null    float64
dtypes: float64(1)
memory usage: 1.4 KB
None
```

DJUchauffage

time	
2014-01-01	324.4
2014-02-01	281.9
2014-03-01	223.9
2014-04-01	135.5
2014-05-01	100.2

```
---  ------       --------------  -----
 0   time         96 non-null     datetime64
 1   DJUclimat    96 non-null     float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 1.6 KB
None
```

DJUclimat

time	
2014-01-01	0.0
2014-02-01	0.0
2014-03-01	0.0
2014-04-01	0.0
2014-05-01	2.2

```
Energy = energy[energy['Territoire']=='Ile-de-France
Energy = Energy.iloc[12:, :]
Energy.set_index("Mois", inplace= True)
#Energy=Energy.rename({'Mois':'time'}, axis=1)
print(Energy.info())
Energy.head()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 95 entries, 2014-01-01 to 2021-11-(
Data columns (total 1 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   consom_total  95 non-null     int64
dtypes: int64(1)
memory usage: 1.5 KB
None
```

consom_total

Mois	
2014-01-01	7612
2014-02-01	6749
2014-03-01	6509
2014-04-01	5396
2014-05-01	5279

* Ici ces 3 jeux de données sont tous pour région **Ile-de-France**
* seuil de référence température pour DJU: **18°C**

1.3 Constitution de dataframe

```
DJU= pd.merge(DJUclimat, DJUchauffage,left_index=True, right_index=True)
DJU.tail()
```

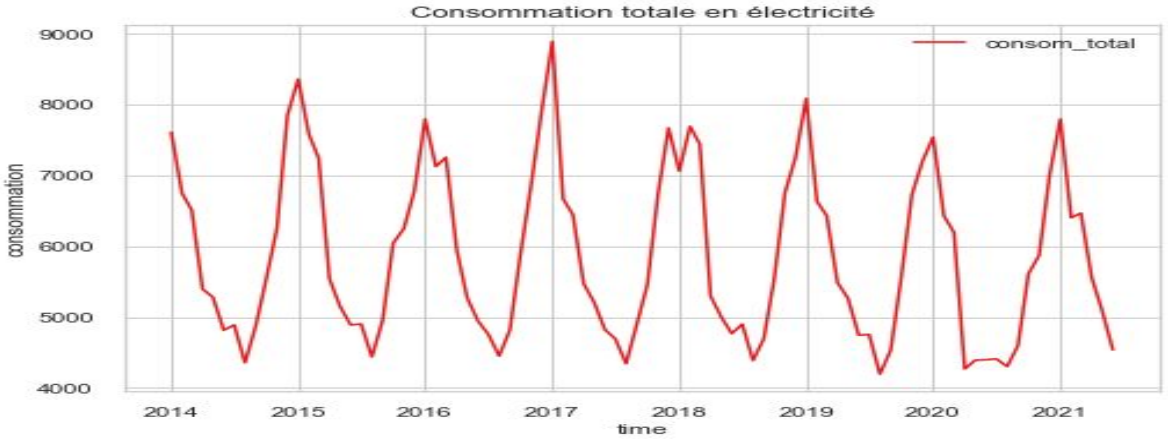
time	DJUclimat	DJUchauffage
2021-02-01	0.0	302.8
2021-03-01	0.2	271.0
2021-04-01	0.9	228.3
2021-05-01	5.1	138.3
2021-06-01	47.6	1.4

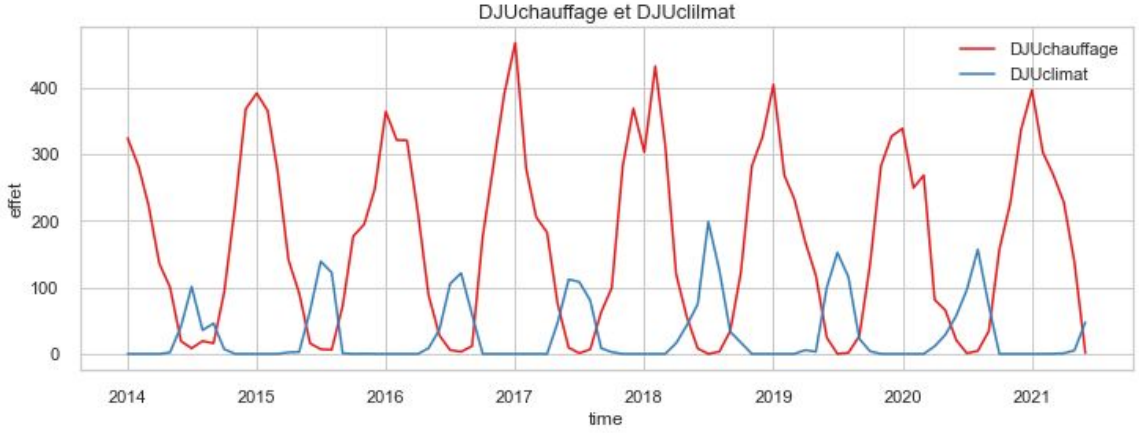
```
df= pd.merge(DJU, Energy, left_index=True, right_index=True)
df.index.names= ['time']
print(df.info())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 90 entries, 2014-01-01 to 2021-06-01
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   DJUclimat    90 non-null     float64
 1   DJUchauffage 90 non-null     float64
 2   consom_total 90 non-null     int64
dtypes: float64(2), int64(1)
memory usage: 2.8 KB
None
```

time	DJUclimat	DJUchauffage	consom_total
2014-01-01	0.0	324.4	7612
2014-02-01	0.0	281.9	6749
2014-03-01	0.0	223.9	6509

1.4 illustration





2. Correction de la consommation de l'effet température due au chauffage électrique

via régression linéaire

2. régression linéaire 1

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from scipy import stats
```

consom_total ~ DJUchauffage

```
reg = smf.ols('consom_total~ DJUchauffage', data=df).fit()
reg.summary()
```

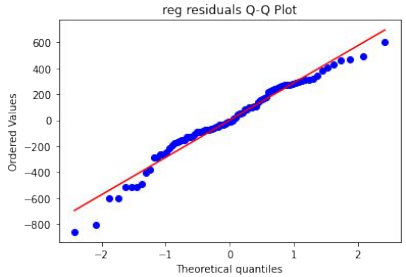
ce model avec DJUchauffage (temperature) comme variable explicative arrive à expliquer **94.3%** de variance en consom_total

OLS Regression Results

Dep. Variable:	consom_total	**R-squared:**	0.943
Model:	OLS	**Adj. R-squared:**	0.942
Method:	Least Squares	**F-statistic:**	1447.
Date:	Wed, 06 Apr 2022	**Prob (F-statistic):**	2.06e-56
Time:	16:11:08	**Log-Likelihood:**	-636.51
No. Observations:	90	**AIC:**	1277.
Df Residuals:	88	**BIC:**	1282.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4438.6175	48.283	91.930	0.000	4342.666	4534.569
DJUchauffage	8.4668	0.223	38.035	0.000	8.024	8.909

P-value < 0.5% => rejet de H0,
DJUchauffage a de l'impact sur consom_total.



distribution des résidus: normalité vérifiée

```
from statsmodels.compat import lzip
import statsmodels

name = ['Lagrange multiplier statistic', 'p-value', 'f-value', 'f p-value']
test = sms.het_breuschpagan(reg.resid, reg.model.exog)
lzip(name, test)
```

```
[('Lagrange multiplier statistic', 0.5205145718556348),
 ('p-value', 0.4706222628074068),
 ('f-value', 0.5119082000094789),
 ('f p-value', 0.47620807233808404)]
```

H0: homoscédasticité des résidus
P-value >5% => non-rejet de H0, ctd. présence de homoscédasticité pour les résidus du modèle reg

2. régression linéaire 2

consom_total ~ DJUchauffage + DJUclimat

```
reg2 = smf.ols('consom_total~ DJUchauffage + DJUclimat', data=df).fit()
reg2.summary()
```

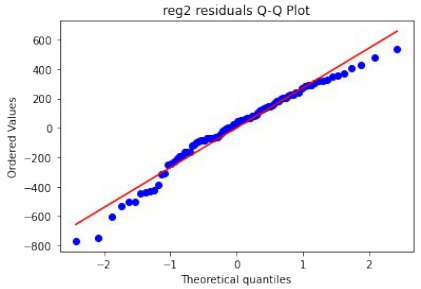
ce model avec arrive à expliquer **94.8%** de variance en consom_total

OLS Regression Results

Dep. Variable:	consom_total	**R-squared:**	0.949
Model:	OLS	**Adj. R-squared:**	0.948
Method:	Least Squares	**F-statistic:**	806.3
Date:	Wed, 06 Apr 2022	**Prob (F-statistic):**	7.07e-57
Time:	16:11:20	**Log-Likelihood:**	-631.40
No. Observations:	90	**AIC:**	1269.
Df Residuals:	87	**BIC:**	1276.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	4240.4917	76.546	55.398	0.000	4088.348	4392.635
DJUchauffage	9.1505	0.299	30.596	0.000	8.556	9.745
DJUclimat	2.8211	0.872	3.234	0.002	1.087	4.555

P-values < 0.5% => rejet de H0,
DJUchauffage et DJUclimat ont tous de l'impact sur consom_total.

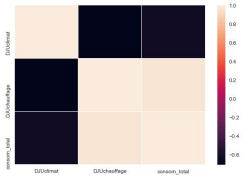


distribution des résidus du reg2: normalité vérifiée

```
name = ['Lagrange multiplier statistic', 'p-value', 'f-value', 'f p-value']
test = sms.het_breuschpagan(reg2.resid, reg2.model.exog)
lzip('reg2', name, test)
```

```
[('r', 'Lagrange multiplier statistic', 0.8346853429817336),
 ('e', 'p-value', 0.6587951309509059),
 ('g', 'f-value', 0.40720780899355646),
 ('2', 'f p-value', 0.666767636817603)]
```

P-value >5% => non-rejet de H0,
homoscédasticité de résidus est vérifiée



corrélation des 2 varaibles explicatives: -50%
reg2 a ses 2 variables explicatives fortement corrélées, et reg2 n'est pas bcp mieux que reg en variance expliquée, donc on pourrait utiliser reg.

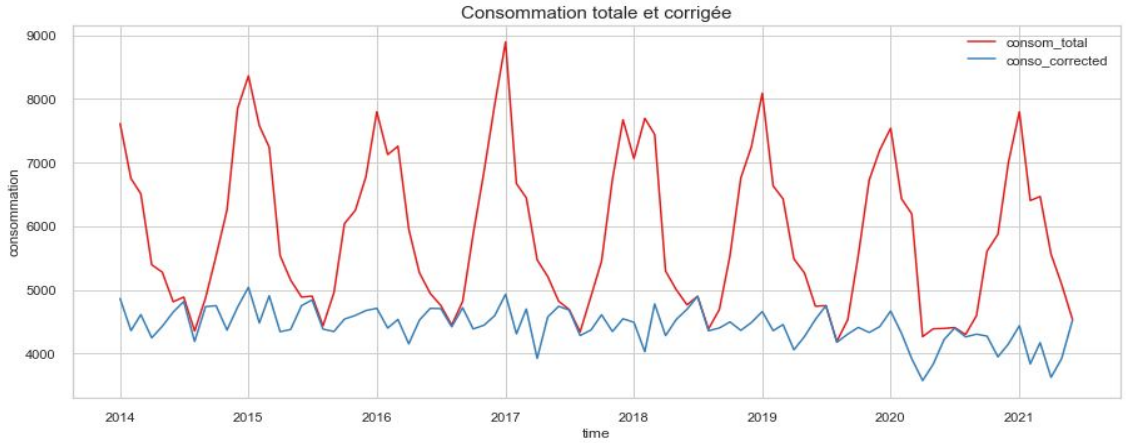
2. Calcul: consommation corrigée de DJUchauffage (reg)

consommation corrigée = consommation totale - DJUchauffage * coeff de rég

```
df['conso_corrected'] = df['consom_total'] - df['DJUchauffage']*reg.params['DJUchauffage']
df.head()
```

*corriger la consommation de l'effet température due au chauffage électrique (DJUchauffage)

time	DJUclimat	DJUchauffage	consom_total	conso_corrected
2014-01-01	0.0	324.4	7612	4865.366090
2014-02-01	0.0	281.9	6749	4362.205613
2014-03-01	0.0	223.9	6509	4613.280726



conso_corrected oscillent entre 4000-5000

3. Désaisonnalisation (Correction de Variation Saisonnière)

de la consommation suite à la correction précédente

3.1 méthode régression linéaire

3.2 méthode moyennes mobiles

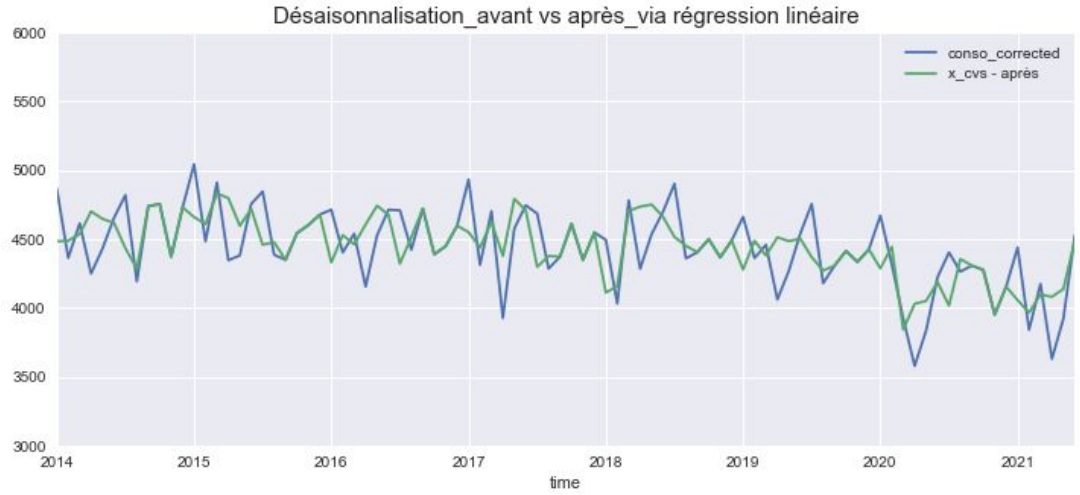
3.1 désaisonnalisation via régression linéaire

data avant désaisonnalisation conso_corrected (x) => série désaisonnalisée (x_cvs)

time	conso_corrected
2014-01-01	4865.366090
2014-02-01	4362.205613
2014-03-01	4613.280726
2014-04-01	4248.746933
2014-05-01	4430.625408

x_cvs

time	
2014-01-01	4482.512574
2014-02-01	4487.617398
2014-03-01	4535.555549
2014-04-01	4699.987306
2014-05-01	4646.091933
2014-06-01	4616.834461
2014-07-01	4432.245913
2014-08-01	4282.979502
2014-09-01	4738.531003



fluctuation x_cvs un peu moins que conso_corrected

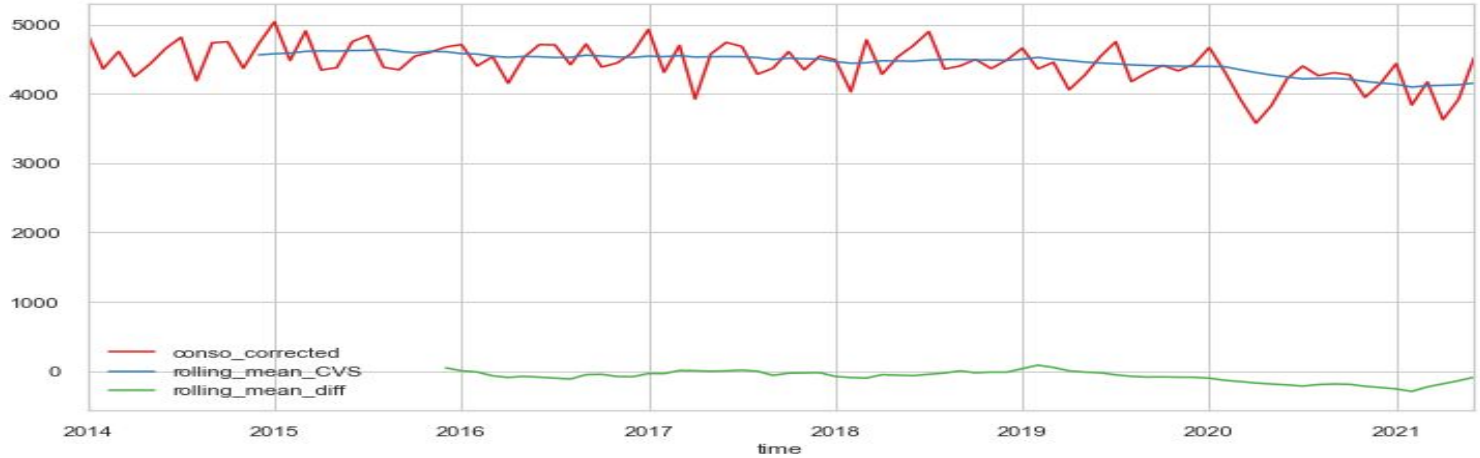
3.2 désaisonnalisation via méthode moyennes mobiles (rolling mean)

```
X= x.copy()
X['rolling_mean_CVS'] = X.rolling(window=12).mean()

X['rolling_mean_diff'] = X['rolling_mean_CVS'] - X['rolling_mean_CVS'].shift(12)
```

*désaisonnalisation => moyennes mobiles.
window=12: Calculer les moyennes à base de tous les 12 observations

*différenciation: difference between current time step & 12 last steps
.shift(12): seasonality changes per 12 months



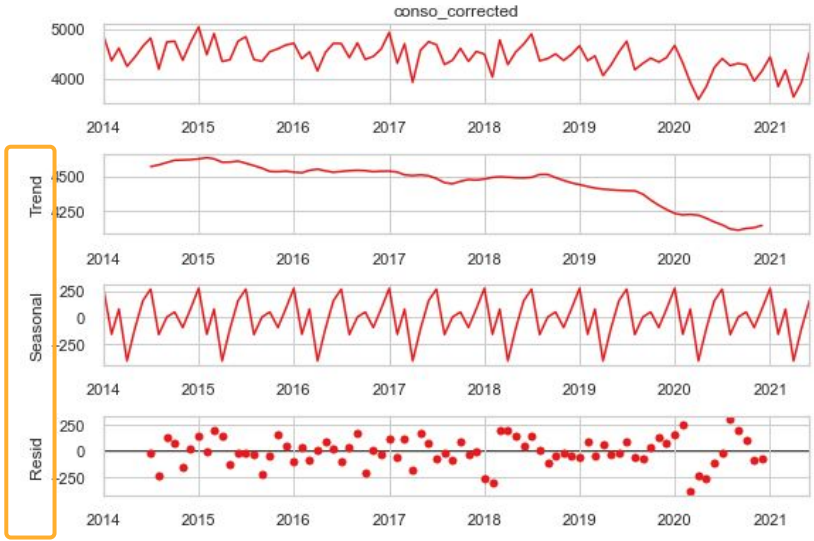
Observations:

- rolling_mean_CVS (moyenne mobile) : tendance, car la saisonnalité de data brut est déduite à travers traitement **X.rolling(window=12).mean()**
- rolling_mean_diff (résidus) : quasiment constante et ses valeurs se concentrent **autour de 0**.
- ce résultat résonne avec celui du statsmodel 'seasonal_decompose'

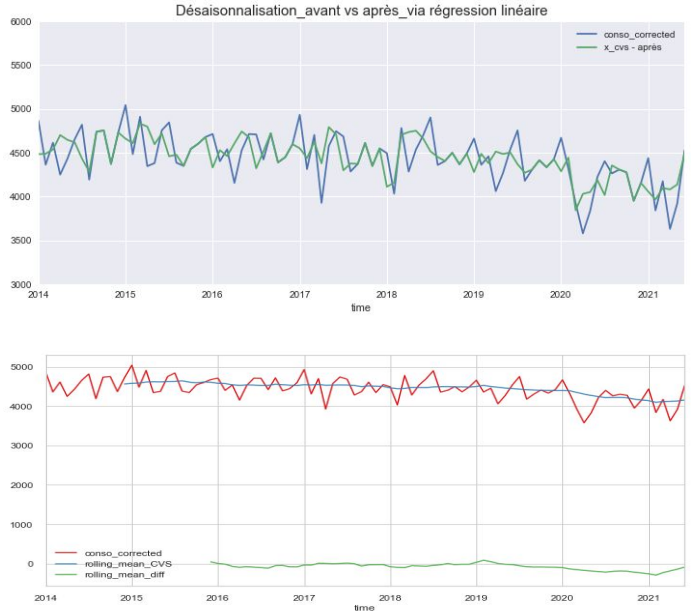
3.3 **désaisonnalisation_comparaison**_algorithme de décomposition

```
decomp_x = seasonal_decompose(df['conso_corrected'], model='add')
```

* on observe dans conso_corrected que l'**ampleur de saisonnalité** ne change pas avec le temps (non-exponentiel),donc on opte pour **'additive'**



* 3 composantes de série temporelle



En référant le résultat de décomposition seasonal_decompose, on voit que la 2ème **méthode (moyennes mobiles)** est plus performante en désaisonnalisation.

4. Prédiction de la consommation en électricité

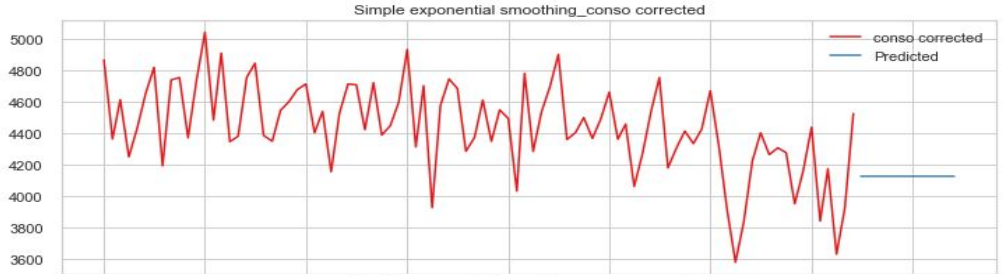
- Lissage exponentiel
- SARIMAX

Respectivement, pour quelle raison on peut utiliser ces deux méthodes ?

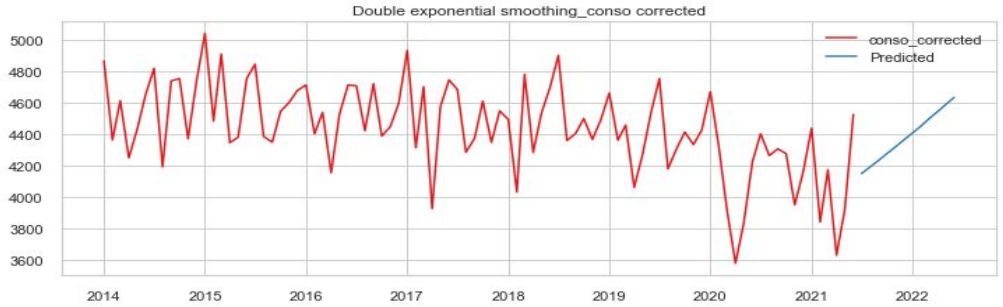
- Lissage exponentiel (Exponential Smoothing) attache plus d'importance sur les données récentes, ctd. elle est plus adaptée à la prédiction pour les séries qui changent vite au fil du temps (météo, CA des start-up, un nouveau produit…)
- A la différence d'**ARMA et ARIMA** qui ne peuvent être appliqués dans série sans saisonnalité, **SARIMA et SARIMAX** permet de modéliser les données qui présentent une saisonnalité.

ARMA: AutoRegressive Integrated Moving Average
ARIMA: AutoRegressive Integrated Moving Average
SARIMA: Seasonal Auto-Regressive Integrated Moving Average
SARIMAX: Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors

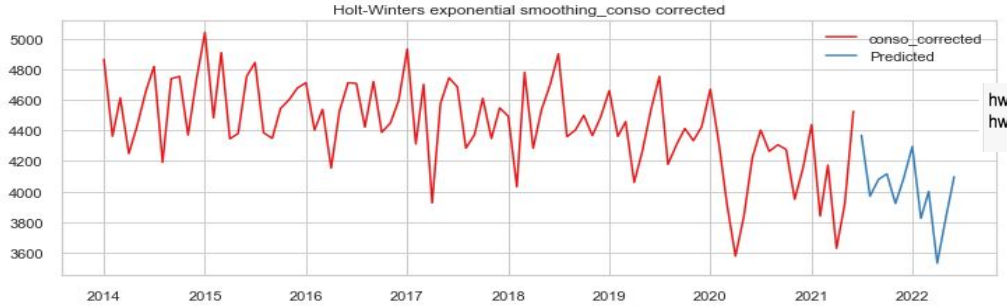
4.1 Lissage exponentiel: simple, double, Holt-Winters



LES: ni la tendance ni la saisonnalité n'est prise en compte



LED: tendance oui, mais saisonnalité non




```
hw = ExponentialSmoothing(np.asarray(X["conso_corrected"]), seasonal_periods=12, trend='add', seasonal='add').fit()
hw_pred = hw.forecast(12)
```

HW: tendance & saisonnalité toutes prises en compte par cette méthode, prédiction a l'air relativement satisfaisante

4.2 SARIMAX_model3 (model choisi)

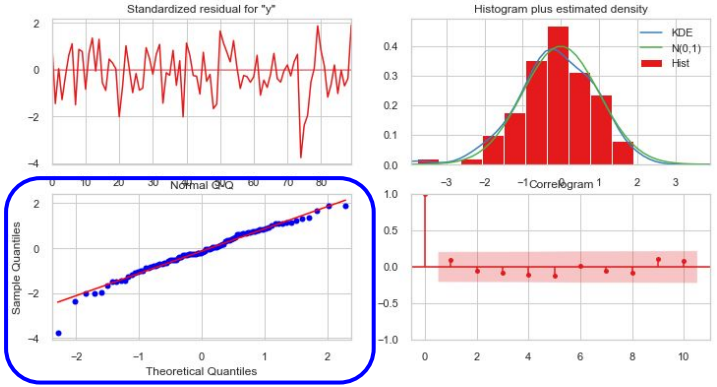
```
model3 = SARIMAX(np.asarray(x['conso_corrected']), order=(1,0,1), seasonal_order=(1,0,1,12))
results3 = model3.fit()          # training
results3.summary()
```

p-values < 5% =< rejet de H0,
model3 n'a pas de paramètres non-significatifs

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9974	0.010	95.477	0.000	0.977	1.018
ma.L1	-0.8055	0.075	-10.812	0.000	-0.952	-0.660
ar.S.L12	0.9963	0.020	50.563	0.000	0.958	1.035
ma.S.L12	-0.8854	0.291	-3.037	0.002	-1.457	-0.314
sigma2	2.597e+04	4436.588	5.853	0.000	1.73e+04	3.47e+04

Ljung-Box (L1) (Q):	0.84	Jarque-Bera (JB):	7.78
Prob(Q):	0.36	Prob(JB):	0.02
Heteroskedasticity (H):	1.55	Skew:	-0.55
Prob(H) (two-sided):	0.24	Kurtosis:	3.93

H0: homoscédasticité (présence de hétéroscédasticité)
non-rejet de H0: **homoscédasticité** de résidu vérifiée



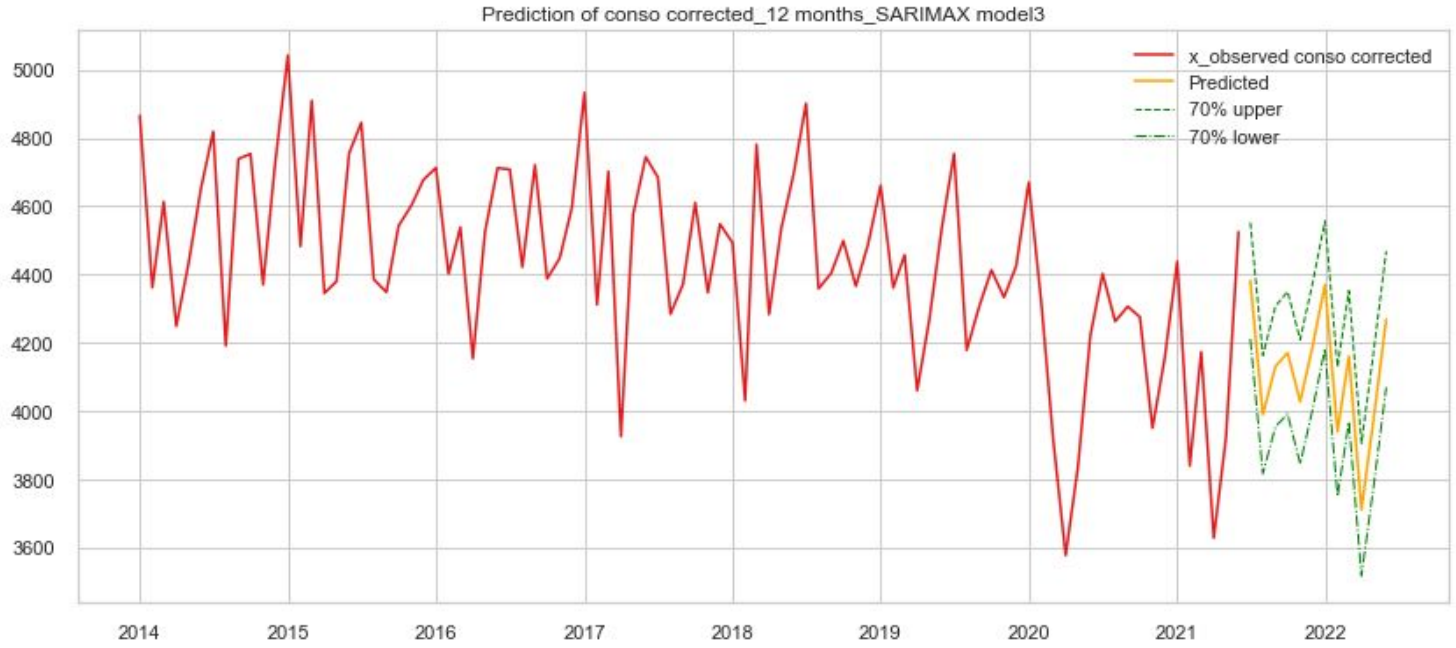
normalité de résidu vérifiée

test de blancheur du résidu: ljungbox test

H0: les valeurs du résidu se distribuent indépendamment, résidu est un bruit blanc sans autocorrélation.

non-rejet de H0: **blancheur** de résidu vérifié.

4.2 SARIMAX_prédiction



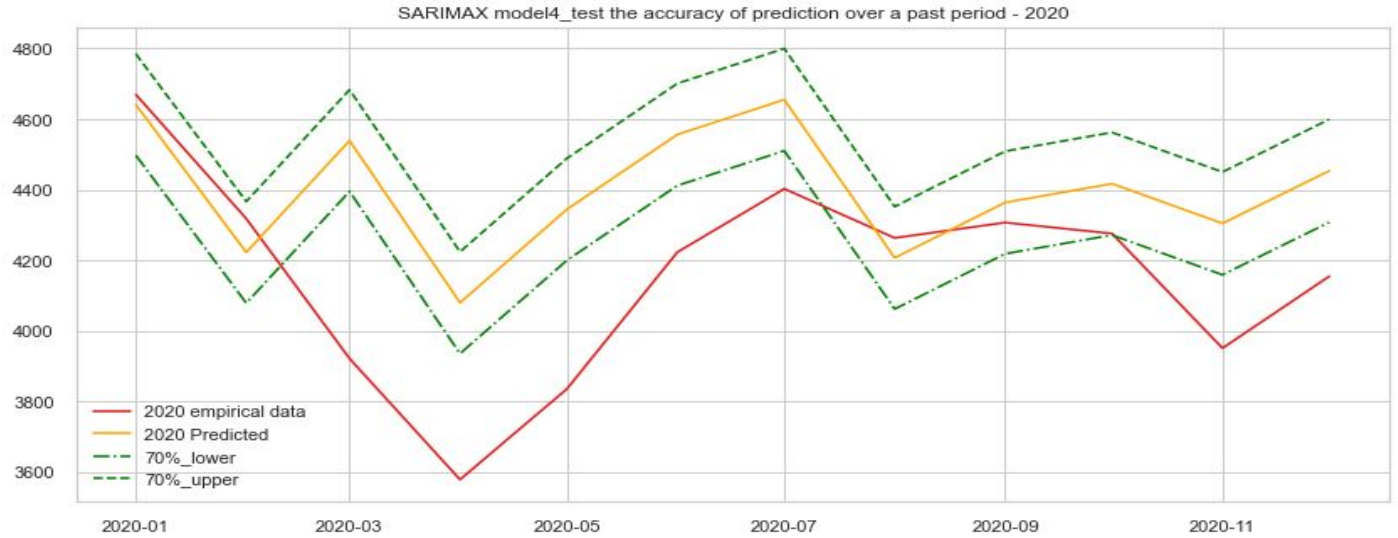
5. Analyse à posteriori

- **test de précision sur une période passée**

Pour vérifier la performance de nos models de deux méthodes (ExponentialSmoothing vs SARIMAX), ici on va faire une analyse comparative des données observées et ‘prédites’ pour une période passée

5.1 SARIMAX_test de précision sur une période passée via **graphique**

```
x_past = x['conso_corrected'][:'2019']
# y_past = np.log(x_past)
x_forecast = x['conso_corrected']['2020']  # ‘prédire’ la consommation en 2020
```



=> Conclusion: en terme de tendance et fluctuation de saisonnalité, ce model a une performance satisfaisante.

5.1 SARIMAX_test de précision sur une période passée via **métriques**

```
# Evaluate accuracy of prediction model for year 2020 - via metrics

from sklearn.metrics import *

print('MAE:', mean_absolute_error(x_forecast, pred_past))          # importance d'une erreur est linéaire avec son amplitude, si le dataset contient des valeurs
print('MSE:', mean_squared_error(x_forecast, pred_past))           # carré, exponentiel. quand on accorde grande importance aux grandes erreurs
print('RMSE:', np.sqrt(mean_squared_error(x_forecast, pred_past))) # Root Mean Squared Error: more reasonable than MSE , additive
print('median abs err:', median_absolute_error(x_forecast, pred_past))    #Median absolute error  = median(|Yvrai - Y predI) : très peu sensible aux grande erre
print('MAPE', mean_absolute_percentage_error(x_forecast, pred_past)*100, '%')      # MAPE:  puts a heavier penalty on negative errors than on positive error
```

```
MAE: 270.5459186289376
MSE: 109924.36755772762
RMSE: 331.54843923283306
median abs err: 275.86205359088945
MAPE 6.802266940024825 %
```

MAE =(1/n) * Σ(|réalité – prédit|)

- [0, infini positif[, moyenne arithmétique des erreurs absolues
- facile à interpréter, peut être grand si dataset contient outliers

$$\mathrm{RMSD}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{\mathrm{E}((\hat{\theta} - \theta)^2)}.$$

- RMSE: [0, infini positif[, c'est **écart-type des résidus** (distance entre la prédiction et la réalité)
- RMSE mesure à quel niveau les données sont concentrées autour de la ligne de best-fit
- et sert à comparer les erreurs de différents modèles prédictifs pour un ensemble de données particulier

MAPE = (1/n) * Σ(|réalité – prédit| / |réalité|) * 100

- [0,1], c'est la **différence moyenne en prédiction et réalité**, bon pour les données non-négatives
- intuitive et facile à interpréter

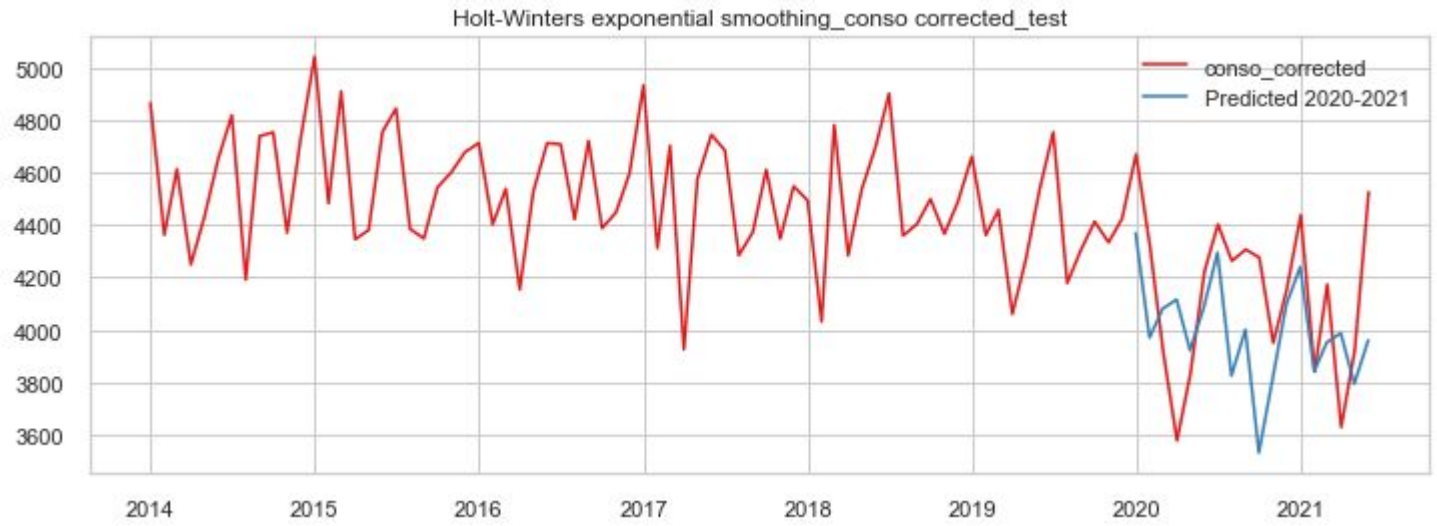
Selon les résultats de ces métriques entre autres RMSE et MAPE, on en conclut que ce model est performant.
niveau de précision SARIMAX model: **93.2 %**

5.2 **Holt-Winters**_test de précision sur une période passée via **graphique**

```
X_test =  X['conso_corrected']
X_past = X_test[:'2019']
X_forecast = X_test['2020':'2021']
print(X_forecast.shape)
```

'prédire' la consommation 2020-2021 (18 mois)

(18,)



Conclusion:

- en terme de tendance, HW est performant. mais au niveau de saisonnalité, HW n'est pas assez satisfaisante.
- Vu l'année 2020 n'est pas assez typique (COVID19 et confinement), on dit que l'anomalie de l'année 2020 a fait l'effet de déviation.

5.2 Holt-Winters_test de précision sur une période passée via **métriques**

```
print('MAE_HW:', mean_absolute_error(X_forecast, hwpast_pred))
print('RMSE_HW:', np.sqrt(mean_squared_error(X_forecast, hwpast_pred)))
print('MAPE_HW', mean_absolute_percentage_error(X_forecast, hwpast_pred)*100, '%')
```

```
MAE_HW: 267.92226967935557
RMSE_HW: 331.15716772004475
MAPE_HW 6.465679977815281 %
```

niveau de précision Holt-Winters: **93.5 %**

5.3 comparaison de 2 méthodes

Lissage exponentiel Holt-Winters

1. Facile à employer
2. 3 paramètres: **trend, seasonal, seasonal_periods**
3. Peut être utilisé pour les séries qui présentent **saisonnalité ou pas**
4. Pertinent pour prédiction qui attache bcp de poids sur la **période récente**
5. En cas d'événement 'black swan' (ie.Covid19, guerre) => peu rigoureuse & précision impactée

SARIMAX

1. Pas simple à employer : démarche compliquée
2. bcp de paramètres **order=(P,D,Q), seasonal_order=(P,D,Q,s)**, qui donne bcp de possibilités de combinaison
3. Exige tester ACF PACF pour le choix de certains paramètres (identification à priori des models potentiels)
4. Exige vérifier les conditions d'utilisation (vérification des models potentiels): normalité résidu, homoscédasticité résidu, blancheur résidu, non présence de paramètre non-significatifs
5. Rigeur
6. De divers résultats de prédiction: niveau moyen, niveau supérieur et inférieur (alpha=)

Q & A

Merci

P9_PRÉDICTION DE CONSOMMATION EN ÉLECTRICITÉ

Data Analyst_Xuefei ZHANG

08 avril 2022