ENERCOOP

Prédiction de la demande d’électricité

OpenClassrooms - Parcours Data Analyst - Projet n°9 - Nicolas Deschamps de Boishebert - 30/05/2019

Présentation des données

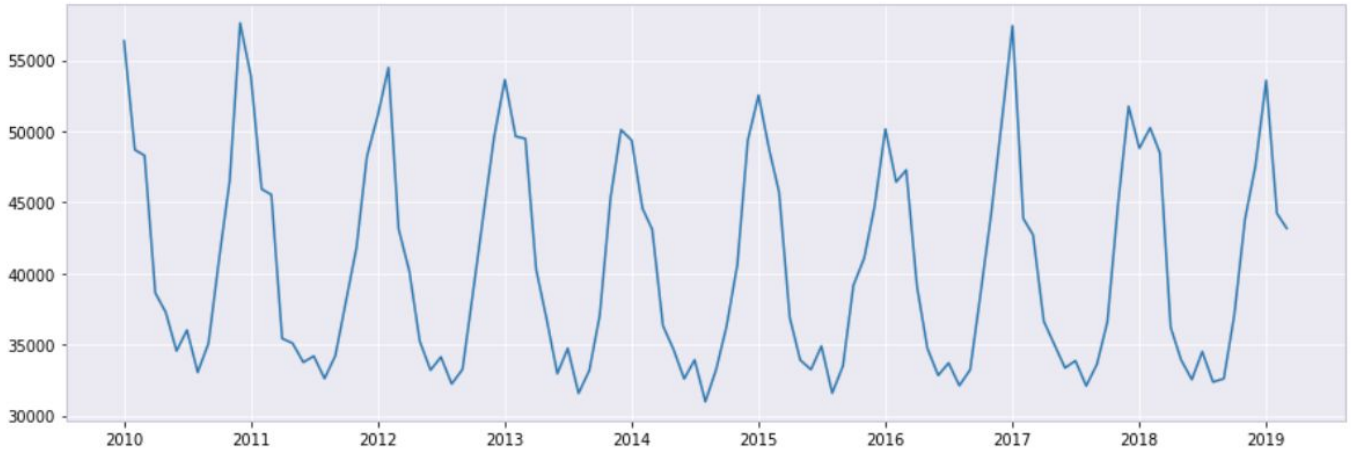
Présentation des données - Consommation de l'électricité en France

Source des données : https://www.rte-france.com/fr/eco2mix/eco2mix-telechargement

```
df = pd.read_csv(
        "data/consommation.csv",
        parse_dates=[0],
        sep=";",
        names=["mois", "pays", "consommation"],
        index_col="mois")
```

```
df.head()
```

mois	consommation
2010-01-01	56342
2010-02-01	48698
2010-03-01	48294
2010-04-01	38637
2010-05-01	37284



Présentation des données - Degrés Jours Unifiés

Source des données : https://cegibat.grdf.fr/simulateur/calcul-dju

Sur un lieu et une période donnés : somme des écarts entre une température de référence (généralement 18 degrés) et la température moyenne journalière (temp. max + temp. min / 2)

Year	1	2	3	4	5	6	7	8	9	10	11	12
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
2010	499.2	371.4	294.5	165.3	140.9	22.6	0.0	11.1	52.3	172.2	310.0	512.0
2011	392.0	304.8	243.1	77.6	43.4	31.4	15.0	11.9	23.2	127.6	226.6	312.7
2012	336.0	435.9	201.9	230.3	83.3	35.0	12.4	2.4	58.0	154.6	296.2	345.9
2013	429.2	402.2	376.6	209.5	158.4	43.6	0.6	5.0	41.5	105.0	303.9	349.5
2014	324.4	281.9	223.9	135.5	100.2	19.1	8.3	19.3	16.0	92.3	222.6	368.2
2015	392.0	365.7	275.5	141.1	91.5	15.8	6.9	6.1	71.9	176.9	195.0	248.1
2016	364.4	321.6	321.1	212.1	88.1	27.5	5.7	3.2	11.7	176.0	285.6	390.8
2017	467.9	278.4	206.1	182.6	75.0	9.4	1.0	6.8	62.6	99.4	282.6	369.0
2018	303.4	432.6	314.3	119.7	55.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Présentation des données - Degrés Jours Unifiés - D'une table à une colonne

From a matrice to a unique column

```
dju = {
    'mois':[],
    'dju':[]
}

# Parse the dataframe and create a new one column based dataframe
for year in df2.index.values:
    for month in df2.columns:
        dju['mois'].append(f"{year}-{month}-01")
        dju['dju'].append(df2.loc[year, month])

dju = pd.DataFrame(dju)
dju['mois'] = pd.to_datetime(dju['mois'])
```

Data merge

```
df3 = df.merge(dju, how="left", left_index=True, right_on="mois")
df3.set_index('mois', inplace=True)
df3.fillna(0, inplace=True)
```

```
df3.head(12)
```

mois	consommation	dju
2010-01-01	56342	499.2
2010-02-01	48698	371.4
2010-03-01	48294	294.5
2010-04-01	38637	165.3
2010-05-01	37284	140.9
2010-06-01	34567	22.6
2010-07-01	36031	0.0
2010-08-01	33069	11.1
2010-09-01	35104	52.3
2010-10-01	40918	172.2
2010-11-01	46532	310.0
2010-12-01	57600	512.0

Présentation des données - Jeu de données final

```
1  df4 = df3[df3.index.year < 2018].copy()
```

Les DJU n'étant pas complets pour 2018, nous travaillerons uniquement sur 2009:2017

Year	1	2	3	4	5	6	7	8	9	10	11	12
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
2010	499.2	371.4	294.5	165.3	140.9	22.6	0.0	11.1	52.3	172.2	310.0	512.0
2011	392.0	304.8	243.1	77.6	43.4	31.4	15.0	11.9	23.2	127.6	226.6	312.7
2012	336.0	435.9	201.9	230.3	83.3	35.0	12.4	2.4	58.0	154.6	296.2	345.9
2013	429.2	402.2	376.6	209.5	158.4	43.6	0.6	5.0	41.5	105.0	303.9	349.5
2014	324.4	281.9	223.9	135.5	100.2	19.1	8.3	19.3	16.0	92.3	222.6	368.2
2015	392.0	365.7	275.5	141.1	91.5	15.8	6.9	6.1	71.9	176.9	195.0	248.1
2016	364.4	321.6	321.1	212.1	88.1	27.5	5.7	3.2	11.7	176.0	285.6	390.8
2017	467.9	278.4	206.1	182.6	75.0	9.4	1.0	6.8	62.6	99.4	282.6	369.0
2018	303.4	432.6	314.3	119.7	55.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Correction de l’effet temperature

Correction de l'effet température - Hypothèse de modèle

$$y = at + bu + \sum_{i=1}^{12} c_i 1$$

Avec :
t qui représente l'évolution temporelle
u qui correspond au DJU du mois en cours
c L'indicatrice du mois en cours

Note : si l'on devait utiliser le modèle dans un but de prédiction et d'estimation de tous les éléments, il faudrait ensuite retrancher la moyenne des valeurs cumulées pour les douze mois et définir cette différence comme ordonnée à l'origine. Mais ici, nous souhaitons simplement connaître le coefficient ***b*** après entraînement du modèle.

Correction de l'effet température - Création d'une indicatrice "mois"

```
df4['month'] = df4.index.month
```

One Hot Encoding

Each column values becomes a new column with "1" or "0". To do that, I use OneHotEncoder from sklearn.preprocessing.

```
# From a column to a 2 dimensions ndarray (n rows, 1 column)
months = df4['month'].values.reshape(-1,1)

# Create a new instance from OneHotEncoder fitted on "month"
onehotencoder = OneHotEncoder(categories="auto", sparse=False).fit(months)

# Get the name of each new column
columns = onehotencoder.categories_[0]

# Values
values = onehotencoder.transform(months)

# Datframe
onehot = pd.DataFrame(values, columns=columns, index=df4.index)

# Merging
df5 = df4.merge(onehot, how="left", left_index=True, right_index=True)

# I don't need the 'month' feature anymore, so I drop it.
df5.drop('month', axis=1, inplace=True)
```

Correction de l'effet température - Création d'une indicatrice "mois"

mois	consommation	dju	1	2	3	4	5	6	7	8	9	10	11	12
2010-01-01	56342	499.2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2010-02-01	48698	371.4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2010-03-01	48294	294.5	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2010-04-01	38637	165.3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2010-05-01	37284	140.9	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2010-06-01	34567	22.6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2010-07-01	36031	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2010-08-01	33069	11.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2010-09-01	35104	52.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2010-10-01	40918	172.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2010-11-01	46532	310.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2010-12-01	57600	512.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

Correction de l’effet température - Création d’une feature “time”

```
1  df5['time'] = np.arange(1,len(df5)+1)
```

```
1  df5.head()
```

mois	consommation	dju	1	2	3	4	5	6	7	8	9	10	11	12	time
2010-01-01	56342	499.2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
2010-02-01	48698	371.4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2
2010-03-01	48294	294.5	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3
2010-04-01	38637	165.3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4
2010-05-01	37284	140.9	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5

Correction de l'effet température - Régression linéaire

```
X = df5.iloc[:,1:]
y = df5.iloc[:,0]
reg_ols1 = sm.OLS(endog = y, exog= X, hasconst=False).fit()
reg_ols1.summary()
```

OLS Regression Results

Dep. Variable:	consommation	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	1.551e+04
Date:	Thu, 30 May 2019	**Prob (F-statistic):**	4.84e-134
Time:	14:44:33	**Log-Likelihood:**	-777.75
No. Observations:	96	**AIC:**	1583.
Df Residuals:	82	**BIC:**	1619.
Df Model:	14		
Covariance Type:	nonrobust		

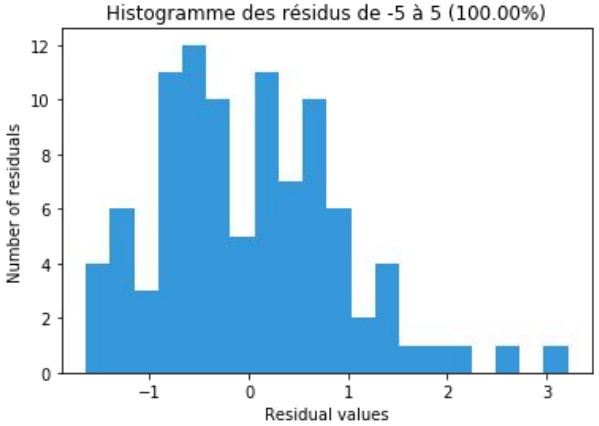
	coef	std err	t	P>|t|	[0.025	0.975]
dju	40.6220	2.122	19.147	0.000	36.401	44.843
1	3.731e+04	933.437	39.971	0.000	3.55e+04	3.92e+04
2	3.432e+04	825.807	41.553	0.000	3.27e+04	3.6e+04
3	3.535e+04	680.243	51.961	0.000	3.4e+04	3.67e+04
4	3.164e+04	510.677	61.958	0.000	3.06e+04	3.27e+04
5	3.198e+04	411.111	77.794	0.000	3.12e+04	3.28e+04
6	3.289e+04	350.769	93.764	0.000	3.22e+04	3.36e+04
7	3.481e+04	345.499	100.739	0.000	3.41e+04	3.55e+04
8	3.234e+04	347.508	93.058	0.000	3.16e+04	3.3e+04
9	3.261e+04	364.905	89.360	0.000	3.19e+04	3.33e+04
10	3.32e+04	471.683	70.390	0.000	3.23e+04	3.41e+04
11	3.35e+04	684.929	48.905	0.000	3.21e+04	3.49e+04
12	3.623e+04	868.815	41.695	0.000	3.45e+04	3.8e+04
time	-12.4987	3.244	-3.852	0.000	-18.953	-6.045

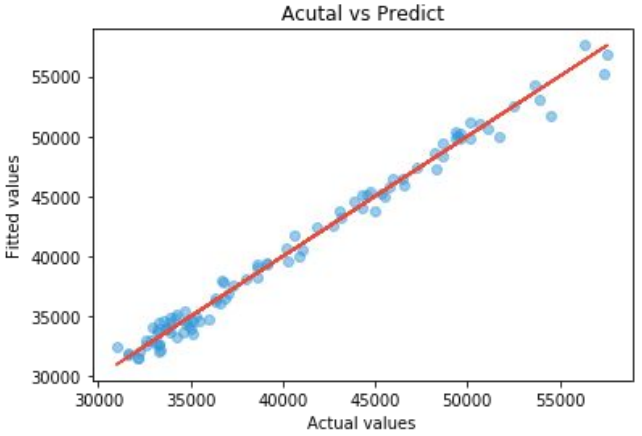
Omnibus:	8.433	**Durbin-Watson:**	1.636
Prob(Omnibus):	0.015	**Jarque-Bera (JB):**	8.057
Skew:	0.648	**Prob(JB):**	0.0178
Kurtosis:	3.580	**Cond. No.**	4.77e+03

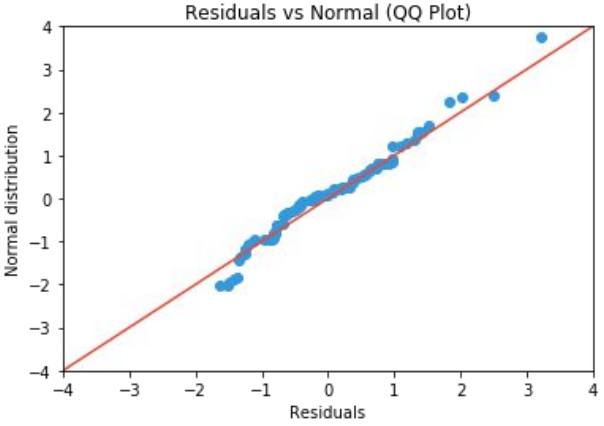
Correction de l’effet température - Régression linéaire - Analyse

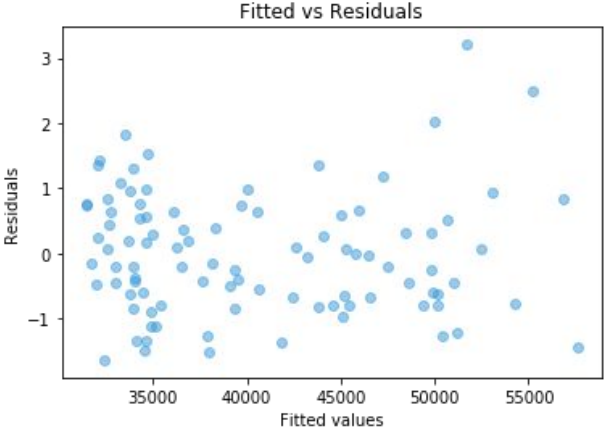
Test de Shapiro :

H0 : normalité des résidus
pvalue : 0.028





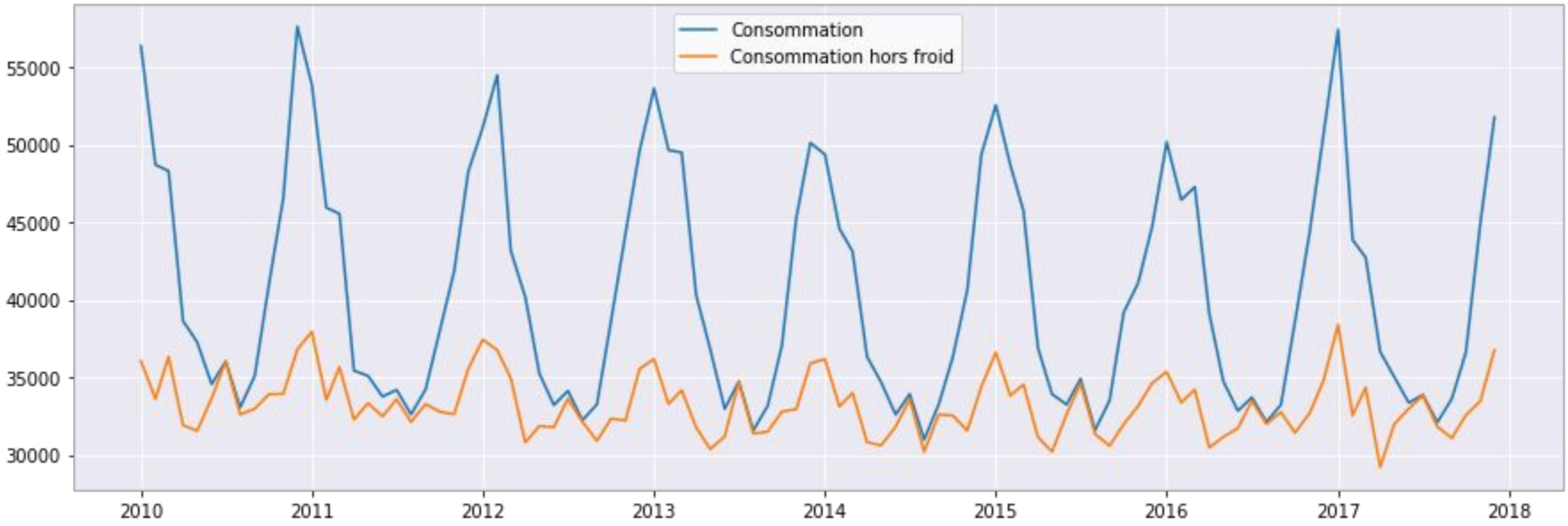




Correction de l'effet température - Résultats

```
# Get the coefficient from the linear regression model
coeff = reg_ols1.params['dju']

# Apply the coeff on DJU column and substract the result to the time series
df5["consommation-hf"] = round(df5["consommation"] - (coeff * df5["dju"])).astype(int)
```



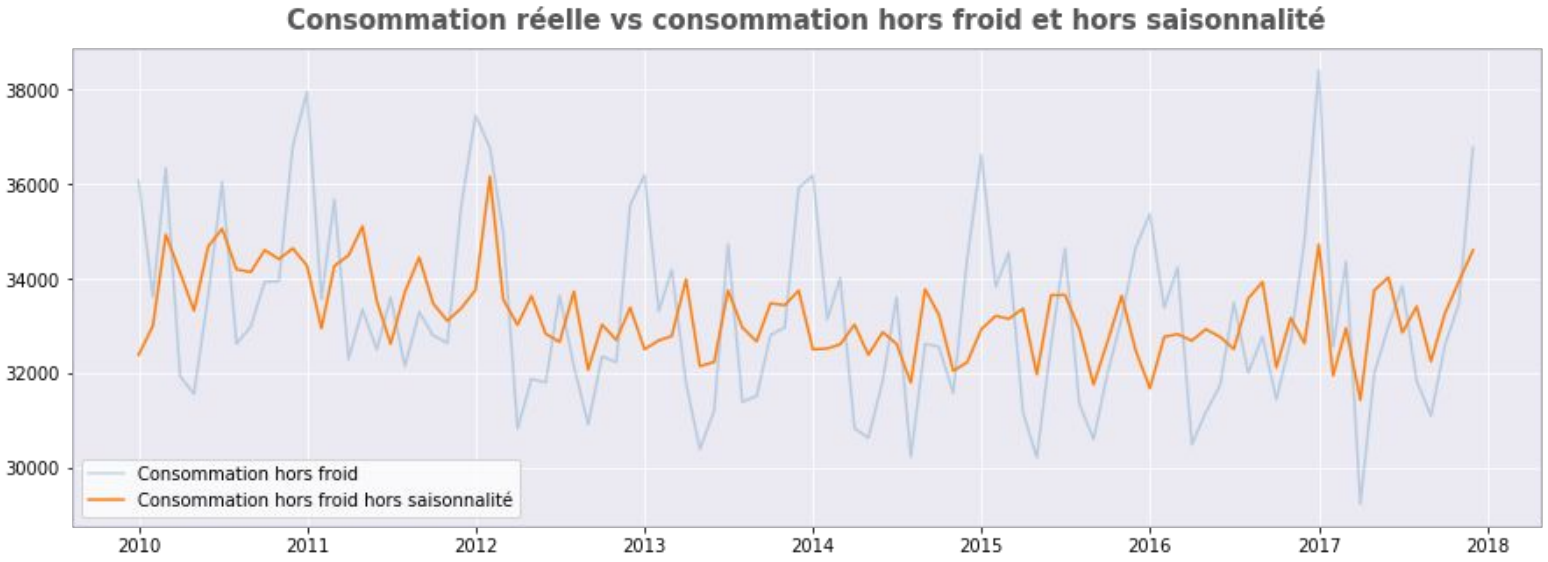
Désaisonnalisation

grâce aux moyennes mobiles

Désaisonnalisation grâce aux moyennes mobiles

J'ai utilisé la méthode **seasonal_decompose()** de StatsModels dont l'algorithme est basé sur les moyennes mobiles.
Documentation : https://www.statsmodels.org/stable/generated/statsmodels.tsa.seasonal.seasonal_decompose.html

```
1  # Seasonal decompose
2  decomp_x = seasonal_decompose(df5["consommation-hf"],  model='additive')
3
4  # Substract the seasonal component
5  df5['consommation-hf-hs'] = df5['consommation-hf'] - decomp_x.seasonal
```



Prévision à l’aide de la méthode Holt-Winters

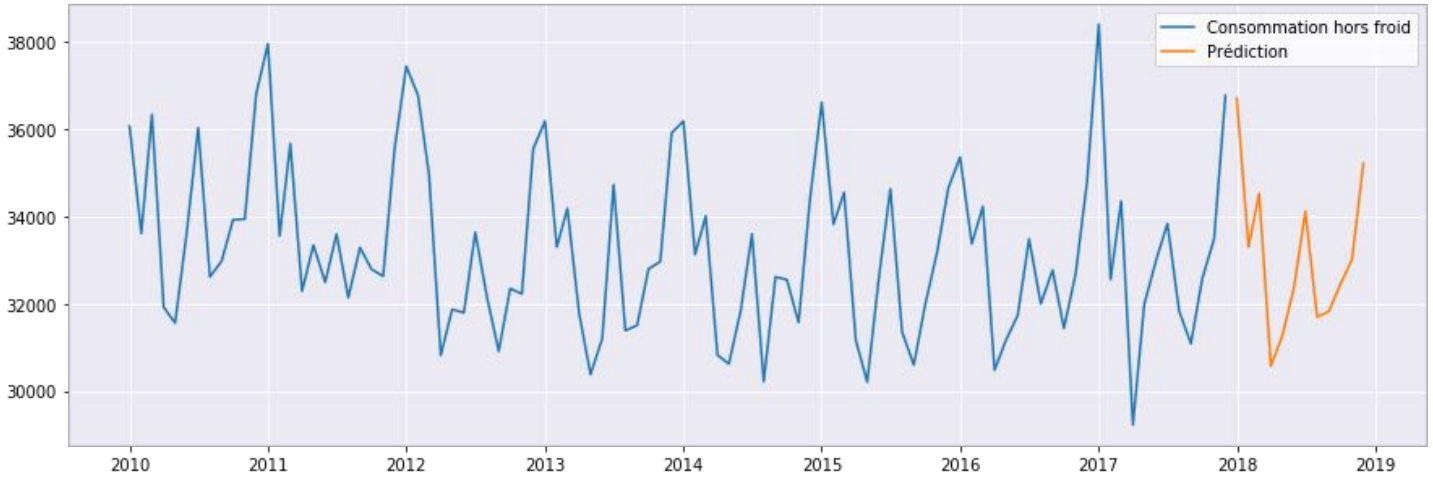
Prévision à l'aide de la méthode Holt-Winters (lissage exponentiel double)

J'ai utilisé la classe **ExponentialSmoothing** de StatsModels.

Documentation : https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html

```
y = np.asarray(df5["consommation-hf"])

hw = ExponentialSmoothing(y, seasonal_periods=12, trend='add', seasonal='add').fit()
hw_pred = hw.forecast(12)
```

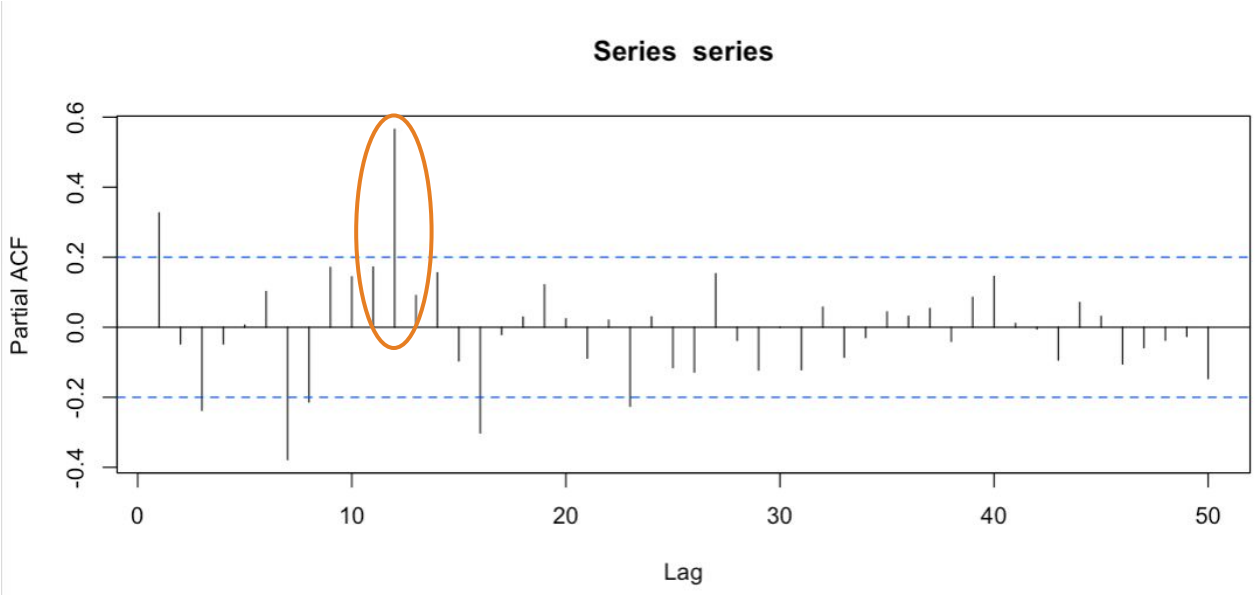


Prévision à l’aide du modèle SARIMA

Seasonal Autoregressive Moving Average

Prévision à l'aide de la méthode SARIMA - Autocorrélogramme partiel

J'ai utilisé la fonction pacf() de R.

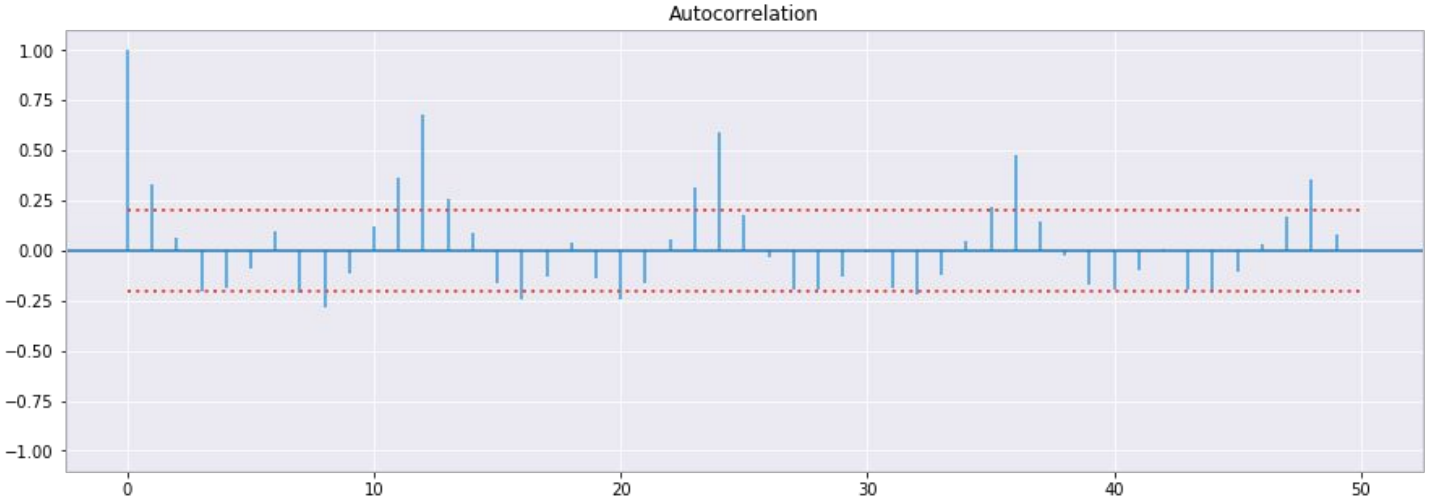


Tendance saisonnière avec un peu d'inertie autour de chaque pic.

Prévision à l'aide de la méthode SARIMA - Autocorrélogramme simple

J'ai utilisé la méthode **plot_acf()** de StatsModels.

Documentation : http://www.statsmodels.org/devel/generated/statsmodels.graphics.tsaplots.plot_acf.html



Tendance saisonnière avec un peu d'inertie autour de chaque pic.

Prévision à l'aide de la méthode SARIMA - Grid Searching

J'ai écrit un algorithme permettant de tester plusieurs modèles et qui calcule pour chaque modèle le RMSE via la méthode du One Step forward consistant à prédire les valeurs une à une en calculant le modèle à chaque fois. En plus du RMSE, je récupère pour chaque modèle :

1. Le test de blancheur des résidus via le test Ljungbox
2. Le test de blancheur des résidus via le test Box-Pierce
3. Le test de normalité des résidus via le test Jarque-Bera
4. AIC (critère d'information d'Akaike)
5. La plus haute pvaleur parmi les paramètres du modèle
6. Le RMSE calculé sur la méthode du One Step Forward

Chaque modèle est calculé avec la classe SARIMAX de statsmodels. Le RMSE est calculé avec la méthode mean_squared_error() de Scikit-Learn

Note :

L'hypothèse nulle des trois premiers tests est la blancheur / normalité.

L'hypothèse nulle des paramètres (point 5.) est la non significativité du paramètre

Les points 1 à 5 sont calculés sur le dernier tour de boucle de l'algorithme, autrement dit sur n-1 valeurs.

Prévision à l'aide de la méthode SARIMA - Grid Searching

L'algorithme intègre la parallélisation du calcul grâce à la classe **Parallel** de la librairie **JobLib**. Ceci m'a permis d'essayer 324 paramétrisations différentes pour Sarima en 15 minutes environ avec un processeur Intel i5 quatre coeurs.

```python
# Generate sarima configs
def sarima_configs():

    configs = []
    ps = [0,1,2]
    ds = [0,1]
    qs = [0,1,2]
    Ps = [0,1,2]
    Ds = [0,1]
    Qs = [0,1,2]
    ms = [12]

    for p in ps:
        for d in ds:
            for q in qs:
                for P in Ps:
                    for D in Ds:
                        for Q in Qs:
                            for m in ms:
                                config = ((p,d,q), (P,D,Q,m))
                                configs.append(config)
    return configs
```

Number of analyzed models

```python
len(sarima_configs())
```

```
324
```

Run the algorithm

```python
%%time
data = df5["consommation-hf"].values
df_par = sarima_grid(data, parallel=True)
```

```
CPU times: user 690 ms, sys: 137 ms, total: 827 ms
Wall time: 15min 22s
```

Prévision à l'aide de la méthode SARIMA - Sélection des modèles

Après avoir fait tourner l'algorithme, je récupère uniquement les modèles dont les paramètres sont tous significatifs et pour lesquels tous les tests de blancheur et de normalité sont OK à un niveau de seuil de 5%.

10 modèles sont retenus

```
df_results = sarima_df(df_par)

# Whiteness, pvalue > 0.05
df_results_filtered = df_results[df_results['Ljung-Box'] > 0.05].copy()
df_results_filtered = df_results_filtered[df_results['Box-Pierce'] > 0.05].copy()

# Gaussian, pvalue > 0.05
df_results_filtered = df_results_filtered[df_results['Jarquebera'] > 0.05].copy()

# Parameter pvalue < 0.05
df_results_filtered = df_results_filtered[df_results['pvalue'] < 0.05].copy()

# Good models, ranged by RMSE
display(df_results_filtered.sort_values(by='RMSE'))
```

	config	RMSE	AIC	Ljung-Box	Box-Pierce	Jarquebera	pvalue
256	((2, 0, 2), (0, 1, 1, 12))	1005.184372	1107.401781	0.473321	0.570125	0.649124	4.499125e-05
9	((0, 0, 0), (1, 1, 0, 12))	1106.185111	1192.163070	0.077319	0.127152	0.061323	7.410470e-06
15	((0, 0, 0), (2, 1, 0, 12))	1124.475804	978.004114	0.116577	0.199793	0.435218	4.888924e-02
135	((1, 0, 1), (1, 1, 0, 12))	1151.622155	1175.327071	0.059703	0.101890	0.674031	3.452416e-02
136	((1, 0, 1), (1, 1, 1, 12))	1166.756530	1139.324095	0.105795	0.164347	0.887819	3.284919e-03
79	((0, 1, 1), (1, 0, 1, 12))	1178.772083	1345.421893	0.147726	0.221350	0.373774	5.792705e-10
210	((1, 1, 2), (2, 0, 0, 12))	1195.170407	1149.745710	0.124954	0.181865	0.934578	3.907018e-03
12	((0, 0, 0), (2, 0, 0, 12))	1204.882415	1190.871080	0.103956	0.142843	0.057914	3.386643e-06
255	((2, 0, 2), (0, 1, 0, 12))	1205.510413	1346.763888	0.188187	0.196473	0.481560	1.234983e-08
192	((1, 1, 1), (2, 0, 0, 12))	1281.755907	1165.225407	0.204332	0.250862	0.853285	1.445863e-02
315	((2, 1, 2), (1, 1, 0, 12))	1293.937419	1139.380776	0.067235	0.080480	0.522912	3.333769e-03
307	((2, 1, 2), (0, 0, 1, 12))	1864.841857	1375.246719	0.321160	0.424392	0.834666	5.967855e-04

Prévision à l'aide de la méthode SARIMA - Sélection du modèle

Après plusieurs essais, j'ai finalement gardé le modèle avec les paramètres suivants :

Sarima(2,0,2)(0,1,1,12)

RMSE	AIC	Ljung-Box	Box-Pierce	Jarquebera	pvalue
3646.791681	1130.891984	0.270569	0.358183	0.796015	3.046049e-03

```
# Initialization
data = df5["consommation-hf"].copy()
order = (2, 0, 2)
seasonal_order = (0, 1, 1, 12)

# Model (thanks to statsmodels)
model = SARIMAX(
    data,
    order=order,
    seasonal_order=seasonal_order,
    enforce_stationarity=False,
    enforce_invertibility=False).fit()

# Summary
model.summary()
```

Statespace Model Results

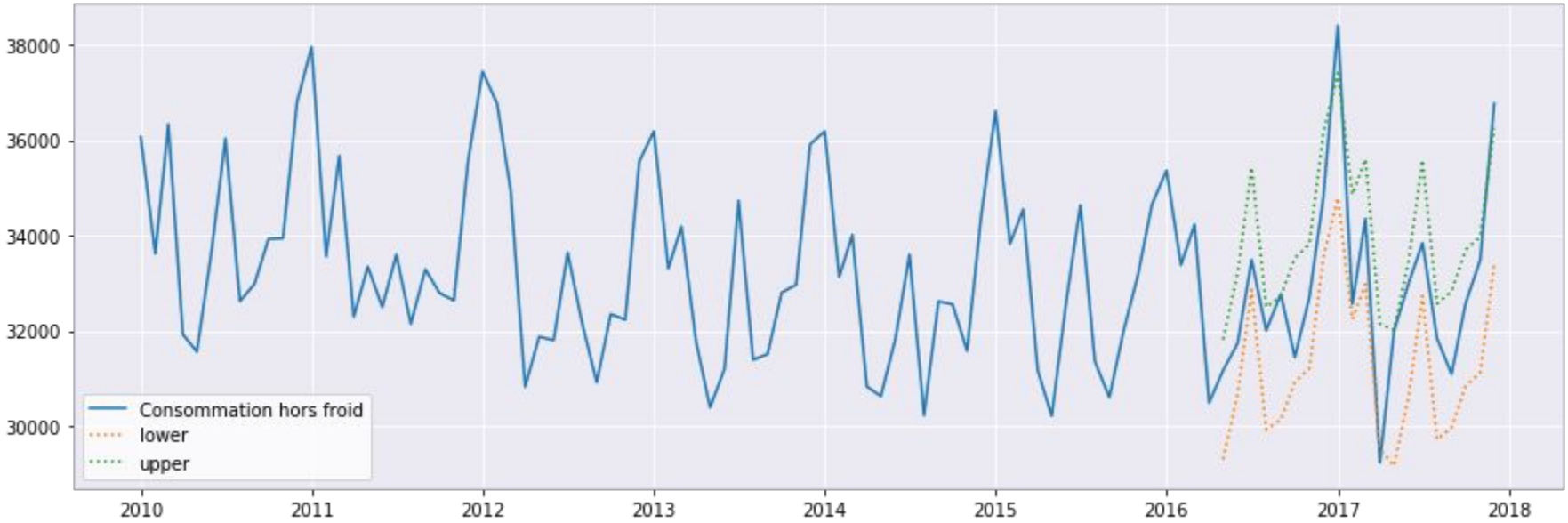
Dep. Variable:	consommation-hf	**No. Observations:**	96
Model:	SARIMAX(2, 0, 2)x(0, 1, 1, 12)	**Log Likelihood**	-559.446
Date:	Mon, 27 May 2019	**AIC**	1130.892
Time:	17:30:20	**BIC**	1144.297
Sample:	01-01-2010	**HQIC**	1136.210
	- 12-01-2017		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.7820	0.081	9.679	0.000	0.624	0.940
ar.L2	-0.8889	0.063	-14.080	0.000	-1.013	-0.765
ma.L1	-1.0850	0.178	-6.081	0.000	-1.435	-0.735
ma.L2	0.9858	0.326	3.028	0.002	0.348	1.624
ma.S.L12	-0.5464	0.068	-8.070	0.000	-0.679	-0.414
sigma2	5.312e+05	1.79e+05	2.963	0.003	1.8e+05	8.83e+05

Ljung-Box (Q):	32.54	**Jarque-Bera (JB):**	0.46
Prob(Q):	0.79	**Prob(JB):**	0.80
Heteroskedasticity (H):	1.06	**Skew:**	0.17
Prob(H) (two-sided):	0.90	**Kurtosis:**	2.78

Prévision à l'aide de la méthode SARIMA - Intervalle de confiance sur les données existantes

On voit que le modèle englobe très bien 2016 / 2017 sur un intervalle de confiance à 95%. Si l'on prend comme valeur le "milieu" de l'intervalle de confiance, cela correspond à la prévision du modèle.



Prévision à l'aide de la méthode SARIMA - Prévision sur un an