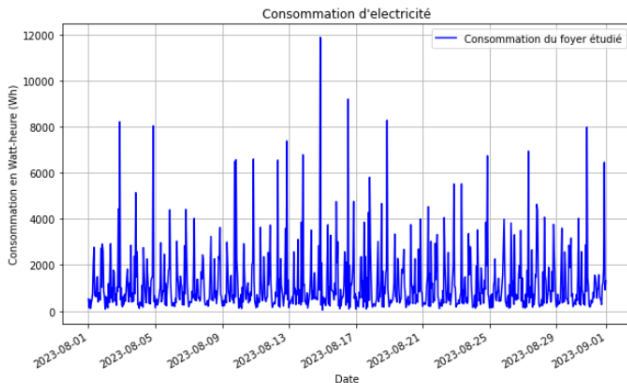


Application des séries chronologiques sur les données Dataset_CB.csv

Ivanhoé & Youness

11 janvier 2024

Introduction



Voici une série chronologique de données pseudo-réelles issue d'une consommation EDF d'un foyer.

Problématique : Comment prédire à l'horizon de 7 jours la consommation électrique du foyer sur la première semaine de septembre 2023 ?

- 1 Partie preprocessing
- 2 Stationnarité et vérification
- 3 Étude des corrélations
- 4 Recherche de modèles significatifs
- 5 Cross validation et évaluation
- 6 Prédiction avec le meilleur modèle
- 7 Conclusion

Renommer les colonnes et passer au format date

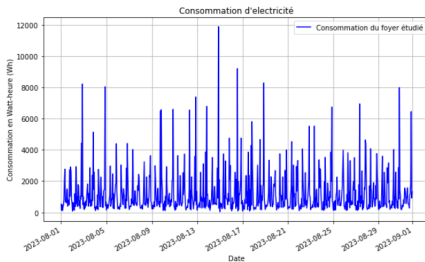
```
EDF = pd.read_csv(path)
EDF.columns = ["DATE", "CONSO"] # renommer col
EDF.DATE = pd.to_datetime(EDF.DATE, format='%d/%m/%y %H:%M')
print("shape = ", EDF.shape)
EDF.head(4)
```

```
shape = (744, 2)
```

Chargement des données

	DATE	CONSO
0	2023-08-01 00:00:00	522.646044
1	2023-08-01 01:00:00	142.889213
2	2023-08-01 02:00:00	497.052422
3	2023-08-01 03:00:00	110.082063

Dataset EDF



Consommation du foyer

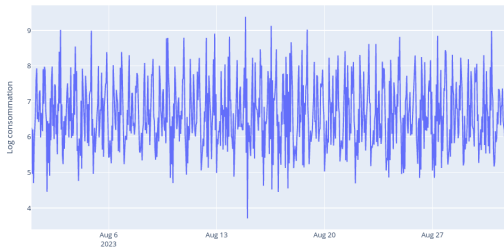
Passage au log pour réduire la variabilité de la série

```
EDF['LOGCONSO'] = np.log(EDF.CONSO)  
EDF.head()
```

	DATE	CONSO	LOGCONSO
0	2023-08-01 00:00:00	522.646044	6.258904
1	2023-08-01 01:00:00	142.889213	4.962070
2	2023-08-01 02:00:00	497.052422	6.208695
3	2023-08-01 03:00:00	110.082063	4.701226
4	2023-08-01 04:00:00	333.036926	5.808253

Dataset EDF

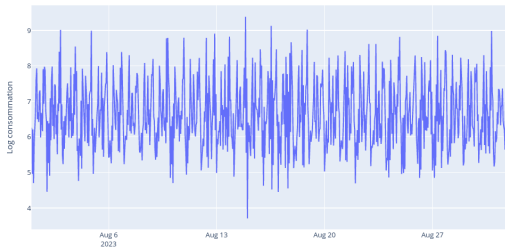
Log-consommation d'énergie du foyer



Log-consommation du foyer

Étude de la stationnarité de la log-consommation

Log-consommation d'énergie du foyer



Log-consommation du foyer

Sans différencier

```
# Testons la non-stationnarité
TestA = adfuller(EDF.LOGCONSO) # Test ADF rejeté
print("ADF p-val : ", TestA[1])

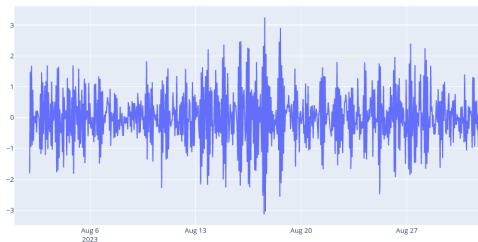
# Testons la stationnarité
TestK = kpss(EDF.LOGCONSO) # Test KPSS non rejeté
print("KPSS p-val : ", TestK[1])# Test non rejeté
```

ADF p-val : 0.0
KPSS p-val : 0.1

Tests ADF et KPSS

Étude de la stationnarité de la log-consommation avec une différenciation saisonnière

Différenciation saisonnière de la Log-consommation d'énergie du foyer



```
# On va différencier une fois pour regarder les incréments à 24 heures près
EDF['diff24'] = EDF.LOGCONSO.diff(24)
fig = px.line(EDF, x='DATE', y='diff24', labels={'DATE': 'Date', 'diff24': ''})
fig.update_layout(title="Différenciation saisonnière de la Log-consommation d'énergie du foyer")
fig.show()
```

Différenciation D=24

Avec différenciation saisonnière s=24

```
# Testons la non-stationnarité
TestA = adfuller(EDF.diff24.dropna()) # Test ADF rejeté au seuil de 5%
print("ADF p-val : ", TestA[1])

# Testons la stationnarité
TestK = kpss(EDF.diff24.dropna()) # Test KPSS non rejeté
print("KPSS p-val : ", TestK[1]) # Test non rejeté

ADF p-val : 0.00022114078230846984
KPSS p-val : 0.1
```

Différentielle saisonnière
log-consommation du foyer

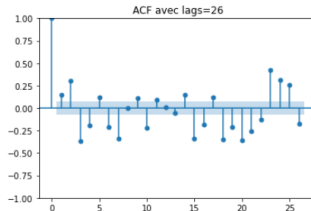
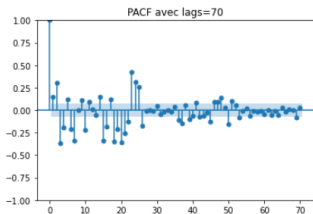
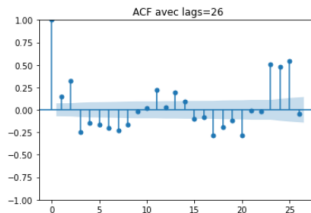
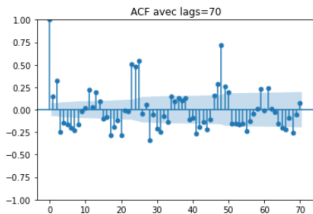
Tests ADF et KPSS

Remarque

Attention avec les tests de stationnarité, l'hétéroscédasticité n'est pas toujours repérée.

Étude des corrélations sur la série passée au log

```
ACF = plot_acf(EDF.LOGCONSO, lags=70, alpha=0.05, title="ACF avec lags=70")  
PACF = plot_pacf(EDF.LOGCONSO, lags=70, alpha=0.05, title="PACF avec lags=70")  
  
ACF = plot_acf(EDF.LOGCONSO, lags=26, alpha=0.05, title="ACF avec lags=26")  
PACF = plot_pacf(EDF.LOGCONSO, lags=26, alpha=0.05, title="ACF avec lags=26")
```



Recherche de modèles à l'aide de la fonction AUTO_ARIMA sous certaines contraintes

```
auto_arma(EDF.LOGCONSO, start_p=0, start_q=0, seasonal=True, start_P=0,  
          start_Q=0, m=24, d=0, D=0, with_intercept=True, stepwise=True, trace=True)  
#force D=d=0 # trace=True pour afficher # stepwise=True passe d'un modèle à l'autre
```

Best model : ARIMA(5,0,0)(1,0,2)[24] intercept

```
auto_arma(EDF.LOGCONSO, start_p=0, start_q=0, max_p = 5,  
          max_q=3, seasonal=True, start_P=0, start_Q=0, m=24, d=1, D=0,  
          with_intercept=True, stepwise=True, trace=True)  
#force d=1 # trace=True pour afficher # stepwise=True passe d'un modèle à l'autre
```

Best model : ARIMA(1,1,0)(1,0,0)[24]

Remarque

Commandes qui mettent beaucoup de temps à s'exécuter.

Modèle 1 : SARIMAX(EDF.LOGCONSO, order=(5, 0, 0), seasonal_order=(1, 0, 2, 24), trend=None)

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT

SARIMAX Results

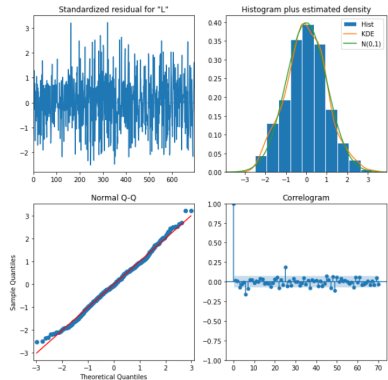
```
Dep. Variable:          LOGCONSO      No. Observations:      744
Model:              SARIMAX(5, 0, 0)x(1, 0, [1, 2], 24)      Log Likelihood      -223.094
Date:                Mon, 08 Jan 2024      AIC      464.188
Time:                15:05:41      BIC      505.083
Sample:              0      HQIC      480.001
```

Covariance Type: opg

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	-0.4553	0.035	-12.916	0.000	-0.524	-0.386
ar.L2	0.3228	0.039	8.222	0.000	0.246	0.400
ar.L3	-0.2471	0.044	-5.648	0.000	-0.333	-0.161
ar.L4	-0.1094	0.041	-2.680	0.007	-0.189	-0.029
ar.L5	0.2526	0.036	6.988	0.000	0.182	0.323
ma.S.L24	0.9999	0.000	8688.332	0.000	1.000	1.000
ma.S.L24	-1.3183	0.133	-9.899	0.000	-1.579	-1.057
ma.S.L48	0.3040	0.065	4.663	0.000	0.176	0.432
sigma2	0.0931	0.013	7.417	0.000	0.068	0.118

```
Ljung-Box (L1) (Q):      0.19      Jarque-Bera (JB):      1.14
Prob(Q):                0.66      Prob(JB):      0.57
Heteroskedasticity (H): 1.45      Skew:      0.10
Prob(H) (two-sided):      0.00      Kurtosis:      2.98
```

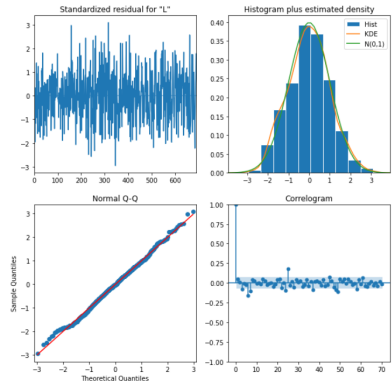
```
plot1 = Mod1.plot_diagnostics(figsize=(10,10),lags=70)
```



Modèle 2 : SARIMAX(EDF.LOGCONSO, order=(5, 0, 0), seasonal_order=(2, 0, 2, 24), trend=None)

```
=====
Dep. Variable:          LOGCONSO      No. Observations:      744
Model:                SARIMAX(5, 0, 0)x(2, 0, [1, 2], 24)      Log Likelihood      -216.498
Date:                  Mon, 08 Jan 2024      AIC      452.996
Time:                  15:11:22      BIC      498.378
Sample:                0      HQIC      470.549
Covariance Type:      0
                        - 744
                        opg
=====
              coef      std err      z      P>|z|      [0.025      0.975]
-----
ar.L1      -0.4634      0.036     -12.723      0.000     -0.535     -0.392
ar.L2       0.2922      0.041      7.124      0.000      0.212      0.373
ar.L3      -0.2793      0.043     -6.479      0.000     -0.364     -0.195
ar.L4      -0.1498      0.039     -3.796      0.000     -0.227     -0.072
ar.L5       0.2477      0.036      6.869      0.000      0.177      0.318
ar.S.L24     0.2925      0.077      3.778      0.000      0.141      0.444
ar.S.L48     0.7074      0.077      9.138      0.000      0.556      0.859
ma.S.L24    -0.5115      0.154     -3.314      0.001     -0.814     -0.209
ma.S.L48    -0.4687      0.126     -3.717      0.000     -0.716     -0.222
sigma2       0.1004      0.015      6.905      0.000      0.072      0.129
=====
Ljung-Box (L1) (Q):                1.58      Jarque-Bera (JB):                1.48
Prob(Q):                           0.21      Prob(JB):                          0.48
Heteroskedasticity (H):             1.46      Skew:                              0.07
Prob(H) (two-sided):               0.00      Kurtosis:                         2.82
=====
```

```
plot2 = Mod2.plot_diagnostics(figsize=(10,10),lags=70)
```



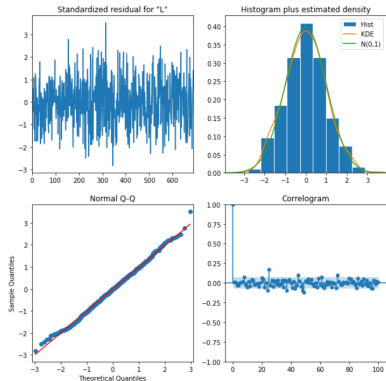
Modèle 3 : SARIMAX(EDF.LOGCONSO, order=(7, 0, 0), seasonal_order=(2, 0, 2, 24), trend=None)

SARIMAX Results

```

Dep. Variable:          LOGCONSO      No. Observations:      744
Model:              SARIMAX(7, 0, 0)x(2, 0, [1, 2], 24)      Log Likelihood      -283.658
Date:                Mon, 08 Jan 2024      AIC      431.316
Time:                15:18:24      BIC      485.739
Sample:              0      HQIC      452.369
Covariance Type:      opg
    
```

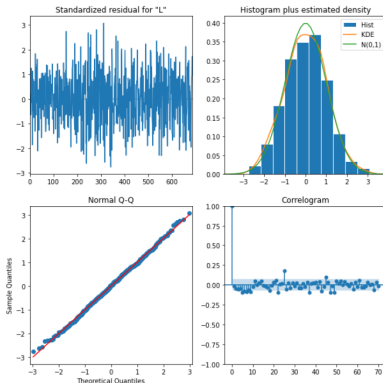
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	-0.4379	0.039	-11.310	0.000	-0.514	-0.362
ar.L2	0.3091	0.042	7.433	0.000	0.228	0.391
ar.L3	-0.3397	0.044	-7.685	0.000	-0.426	-0.253
ar.L4	-0.1595	0.042	-3.816	0.000	-0.241	-0.078
ar.L5	0.2591	0.041	6.333	0.000	0.179	0.339
ar.L6	-0.2088	0.043	-4.836	0.000	-0.293	-0.124
ar.L7	-0.1966	0.040	-4.898	0.000	-0.275	-0.118
ar.S.L24	0.2652	0.069	3.832	0.000	0.130	0.401
ar.S.L48	0.7347	0.069	10.617	0.000	0.599	0.870
ma.S.L24	-0.4982	0.118	-4.216	0.000	-0.730	-0.267
ma.S.L48	-0.4711	0.107	-4.389	0.000	-0.681	-0.261
sigma2	0.0985	0.011	9.081	0.000	0.077	0.120
Ljung-Box (L1) (Q):			0.13	Jarque-Bera (JB):	1.27	
Prob(Q):			0.72	Prob(JB):	0.53	
Heteroskedasticity (H):			1.34	Skew:	0.08	
Prob(H) (two-sided):			0.03	Kurtosis:	2.87	



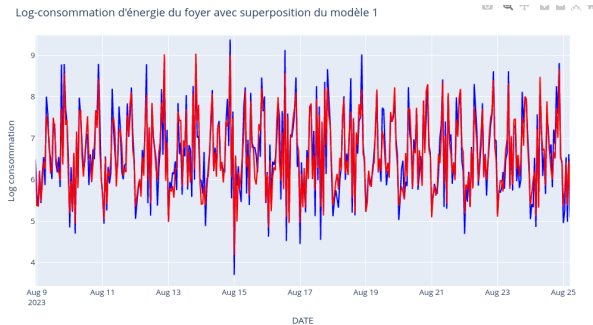
Modèle 4 : SARIMAX(EDF.LOGCONSO, order=(8, 1, 0), seasonal_order=(2, 0, 1, 24), trend=None)

SARIMAX Results

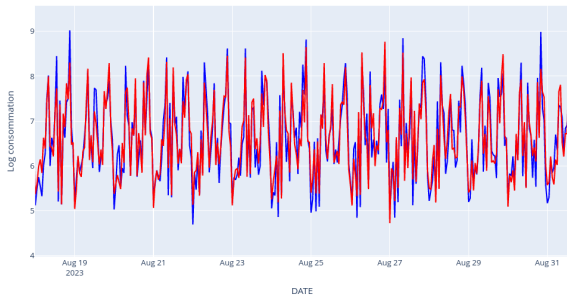
Dep. Variable:	LOGCONSO			No. Observations:	744	
Model:	SARIMAX(8, 1, 0)x(2, 0, [1], 24)			Log Likelihood	-244.235	
Date:	Mon, 08 Jan 2024			AIC	512.470	
Time:	16:03:41			BIC	566.858	
Sample:	0			HQIC	533.512	
	- 744					
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	-1.2711	0.037	-34.257	0.000	-1.344	-1.198
ar.L2	-0.7181	0.056	-12.884	0.000	-0.827	-0.609
ar.L3	-0.9057	0.060	-15.204	0.000	-1.022	-0.789
ar.L4	-0.8994	0.070	-12.803	0.000	-1.037	-0.762
ar.L5	-0.3917	0.071	-5.492	0.000	-0.531	-0.252
ar.L6	-0.4111	0.063	-6.574	0.000	-0.534	-0.288
ar.L7	-0.4329	0.061	-7.136	0.000	-0.552	-0.314
ar.L8	-0.1747	0.039	-4.494	0.000	-0.251	-0.099
ar.S.L24	0.6477	0.035	18.290	0.000	0.578	0.717
ar.S.L48	0.3538	0.035	10.010	0.000	0.285	0.423
ma.S.L24	-0.9914	0.232	-4.271	0.000	-1.446	-0.537
sigma2	0.1055	0.024	4.415	0.000	0.059	0.152
Ljung-Box (L1) (Q):	0.13			Jarque-Bera (JB):	0.77	
Prob(Q):	0.72			Prob(JB):	0.68	
Heteroskedasticity (H):	1.35			Skew:	0.01	
Prob(H) (two-sided):	0.03			Kurtosis:	2.84	



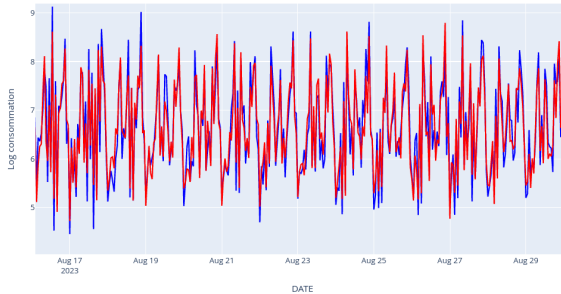
Log-consommation d'énergie du foyer avec superposition du modèle 1



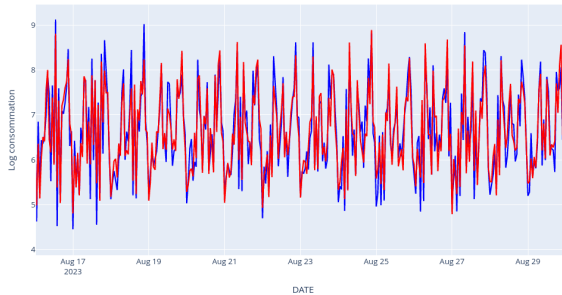
Log-consommation d'énergie du foyer avec superposition du modèle 2



Log-consommation d'énergie du foyer avec superposition du modèle 3



Log-consommation d'énergie du foyer avec superposition du modèle d2



Cross validation : une partie train et une partie test, entraînement sur les données train

Train_test_split

```
M = len(EDF) - int(np.floor(0.2*len(EDF)))
print(M)
#Construire une partie train et une partie test avec 20% des données
EDF_Train = EDF.iloc[:M]
EDF_Test = EDF.iloc[M:]
```

596

Test dataset correspondant à 20% des données

Entraînement de 4 modèles choisis

```
Mod1 = SARIMAX(EDF_Train.LOGCONSO, order=(5, 0, 0), seasonal_order=(1, 0, 2, 24),
               enforce_stationarity=False, enforce_invertibility=False)
Mod1 = Mod1.fit()

Mod2 = SARIMAX(EDF_Train.LOGCONSO, order=(5, 0, 0), seasonal_order=(2, 0, 2, 24),
               enforce_stationarity=False, enforce_invertibility=False)
Mod2 = Mod2.fit()

Mod3 = SARIMAX(EDF_Train.LOGCONSO, order=(7, 0, 0), seasonal_order=(2, 0, 2, 24),
               enforce_stationarity=False, enforce_invertibility=False)
Mod3 = Mod3.fit()

Mod4 = SARIMAX(EDF_Train.LOGCONSO, order=(8, 1, 0), seasonal_order=(2, 0, 1, 24),
               enforce_stationarity=False, enforce_invertibility=False)
Mod4 = Mod4.fit()

## Stocker les modèles dans un dictionnaire
Mod = {"Mod1":Mod1,"Mod2":Mod2,"Mod3":Mod3,"Mod4":Mod4 }
```


Prédiction sur la partie test et évaluation avec certaines métriques

```
def mean_absolute_percentage_error(y_true, y_pred):  
    """Renvoie un pourcentage moyen d'erreur entre les vraies valeurs et les prédictions """  
    y_true, y_pred = np.array(y_true), np.array(y_pred) #convertir au format array  
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100  
  
MAE = [] ## l'écart absolue moyen  
MSE = [] ## la norme2 au carré moyenne  
RMSE = []  
MAPE = []  
  
for i in range(1,5):  
    """Rajoute une colonne prédiction pour chaque modèle et stock les métriques de validations"""  
  
    EDF_Test[f"val_{i}"] = Mod[f"Mod{i}"].get_forecast(steps=len(EDF_Test)).summary_frame(alpha=0.05)['mean']  
    #mean représente la meilleure estimation de chaque période  
    MAE.append(mean_absolute_error(EDF_Test.LOGCONSO, EDF_Test[f"val_{i}"]))  
    MSE.append(mean_squared_error(EDF_Test.LOGCONSO, EDF_Test[f"val_{i}"]))  
    RMSE.append(mean_squared_error(EDF_Test.LOGCONSO, EDF_Test[f"val_{i}"], squared=False))  
    MAPE.append(mean_absolute_percentage_error(EDF_Test.LOGCONSO, EDF_Test[f"val_{i}"]))
```

Dataset test complété avec les prédictions sur la partie test "Forecast_test"

	DATE	CONSO	LOGCONSO	val_exp1	val_exp2	val_exp3	val_exp4	val_1	val_2	val_3	val_4
596	2023-08-25 20:00:00	2789.791163	7.933722	1624.389878	1629.588951	1665.508527	1614.771569	7.346593	7.348657	7.373237	7.335164
597	2023-08-25 21:00:00	3990.209681	8.291599	5233.517739	5005.995358	5020.073489	5037.519473	8.516545	8.470966	8.476551	8.472884
598	2023-08-25 22:00:00	1947.497008	7.574300	819.634774	850.045620	843.339294	856.909432	6.662565	6.697864	6.692721	6.701547
599	2023-08-25 23:00:00	420.785554	6.042123	756.884070	718.223462	663.115417	705.827171	6.582916	6.529355	6.452301	6.507585
600	2023-08-26 00:00:00	294.629097	5.685717	187.620215	176.405297	183.159721	194.259237	5.188126	5.125359	5.165710	5.217408

Remarque

Le meilleur modèle sera celui qui minimise les métriques telles que la MAE et la MSE dans notre cas. En réalité, une métrique différente pourrait être choisie selon la demande d'un métier spécialiste sur la consommation d'électricité.

Comparaison des métriques pour l'évaluation des différents modèles

Métriques de comparaison pour les différents modèles sur les vraies données

Metrics	Models				
	Modèle 1	Modèle 2	Modèle 3	Modèle 4	
	MAE	520.01	499.89	495.51	514.64
	MSE	815369.51	781424.94	773420.09	801150.95
RMSE	902.98	883.98	879.44	895.07	
MAPE	48.93	45.18	44.50	47.50	

Métriques de comparaison pour les différents modèles sur les données au log

Metrics	Models				
	Modèle 1	Modèle 2	Modèle 3	Modèle 4	
	MAE	0.43	0.40	0.40	0.42
	MSE	0.28	0.25	0.24	0.26
RMSE	0.53	0.50	0.49	0.51	
MAPE	6.52	6.12	6.05	6.34	

Prédiction sur la partie test et évaluation avec certaines métriques

```
pred_dates = pd.date_range(start='2023-09-01 00:00:00', periods=7*24, freq='H') #Le temps futur
predictions = np.exp(Pred['mean'].values + Best_Mod.params["sigma2"]/2) # les prédictions à venir

# Créer le graphique Plotly
fig = px.line(EDF, x='DATE', y='CONSO',
              labels={'CONSO': "Consommation d'électricité"})

# Ajouter les prédictions
fig.add_scatter(x=pred_dates, y=predictions, mode='lines',
               name='Prédictions', line=dict(color='green', dash='dot'))

# Ajouter la zone de confiance

##Zone supérieure
fig.add_scatter(x=pred_dates, y=np.exp(Pred['mean_ci_lower']), fill='tonexty',
               mode='lines', line=dict(color='lightgrey'), showlegend=False)

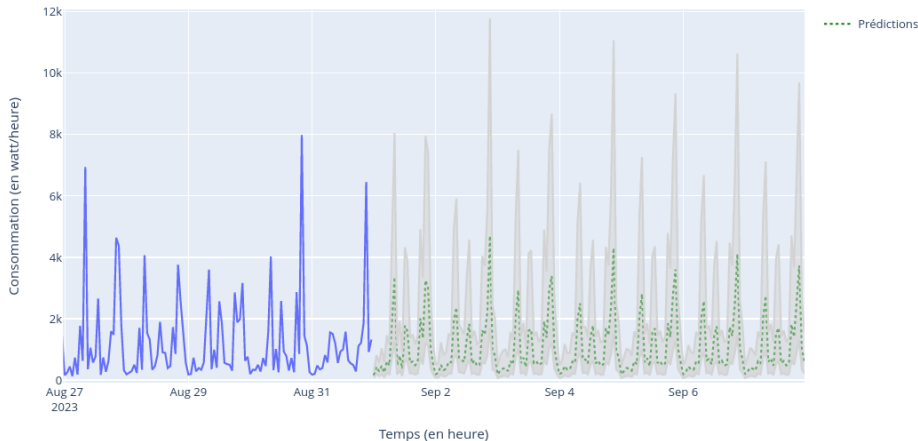
##Zone inférieure
fig.add_scatter(x=pred_dates, y=np.exp(Pred['mean_ci_upper']),
               fill='tonexty', mode='lines', line=dict(color='lightgrey'), showlegend=False)

# Mettre à jour le layout
fig.update_layout(title="Consommation d'électricité avec prédictions",
                  xaxis_title='Temps (en heure)',
                  yaxis_title='Consommation (en watt/heure)')

# Afficher le graphique
fig.show()
```

Représentation graphique des prédictions avec les intervalles de confiance à 95%

Consommation d'électricité avec prédictions



Conclusion

- 1 La RMSE est relativement importante et l'indicateur MAPE sur les données réelle n'est pas non plus très bon (40% d'erreur en moyenne).
- 2 Le traitement des températures, la nébulosité ainsi que la pression atmosphérique pourraient être intégrés dans notre modèle.
- 3 Trouver le caractère saisonnier avec une méthode plus précise comme la transformée de Fourier.
- 4 Tester d'autres méthodes pour traiter les données (ex : réseau de neurones RNN)