MINISTÈRE
DE L'ENSEIGNEMENT
SUPÉRIEUR
ET DE LA RECHERCHE

UPEC
Connaissance - Action

UNIVERSITÉ
PARIS-EST CRÉTEIL
VAL DE MARNE

UNIVERSITE PARIS-EST CRETEIL

Master :

METHODES APPLIQUEES DE LA STATISTIQUE ET DE L'ECONOMETRIE POUR LA RECHERCHE, L'ANALYSE ET LE TRAITEMENT DE L'INFORMATION

Module :

MEMOIRE

APPROCHE PROBABILISTE DE LA PREVISION DE LA CONSOMMATION D'ENERGIE A L'AIDE D'UN RESEAU BAYESIEN DYNAMIQUE

Présenté et soutenu par :

LAJOIE BENGONE AKOU

Sous la supervision de :

SYLVAIN CHAREYRON

ENSEIGNANT CHERCHEUR

Année universitaire :

2024-2025

REMERCIEMENTS

Avant toute chose, je souhaiterai remercier très chaleureusement Monsieur Sylvain Chareyron qui m'a permis de réaliser cette étude. Malgré ses nombreuses occupations, il s'est toujours montré disponible lorsque j'avais besoin de son aide. Sa franchise et ses conseils ont fortement influencé l'orientation de ce projet. Je pense tout particulièrement aux suggestions par rapport à l'introduction et à l'idée de comparer les modèles de prévision.

Aussi, tout au long de ce travail, de nombreuses personnes ont manifesté de l'intérêt pour cette thématique. Je pense principalement à Warren, Léodia , Josué, Aurel, Gomez et Lévy avec lesquels j'ai entretenu des échanges qui m'ont permis de mieux appréhender les enjeux de ce mémoire.

Enfin, je remercie tous mes proches qui m'ont toujours soutenu et accompagné. Il s'agit de ma Famille, mes Amis, mes Camarades et mes Collègues de travail.

TABLE DES MATIERES

Introduction

L'évolution de l'humanité a été permise par une succession de découvertes. Parmi elles figure la maîtrise des énergies. Ces énergies, qui se manifestent sous différentes formes, ont contribué de façon significative à l'amélioration des conditions de vie des individus et au dynamisme des sociétés actuelles.

En effet, de la gestion du patrimoine routier à l'interopérabilité du système bancaire, en passant par la maintenance des réseaux de télécommunication, le fonctionnement des infrastructures modernes est assuré en grande partie par l'intervention des énergies.

Cependant, la disponibilité de ces énergies tend de plus en plus à diminuer, menaçant ainsi les modes de production et de consommation actuels. Les causes apparentes de ce phénomène sont essentiellement d'ordre démographique et climatique. Parmi elles, on retrouve la croissance de la population mondiale et la raréfaction des ressources naturelles, qui créent des tensions sur le marché de l'énergie en matière d'approvisionnement, de distribution et de consommation. Ces tensions, si elles ne sont pas maîtrisées, peuvent entraîner des pertes financières aussi bien chez les consommateurs que chez les gestionnaires d'énergie.

Cela a notamment été le cas pour les consommateurs européens, qui, en 2022, ont été affectés par l'augmentation des coûts de l'électricité. Il en résulte une baisse du pouvoir d'achat pour les ménages et une perte de compétitivité pour les industriels (International Energy Agency, 2023).

Ainsi, face à cette crise énergétique, plusieurs travaux ont été menés dans le but d'utiliser de façon plus efficiente l'énergie disponible. Parmi eux, on retrouve le développement de modèles de prévision de la consommation d'énergie à partir des méthodes d'analyse de données. Ces modèles de prévision bénéficient aujourd'hui d'une grande attention de la part de la communauté scientifique, qui voit en eux un moyen d'estimer la demande future d'énergie. Cette demande estimée, si elle s'avère correcte et parfaitement ajustée à l'offre d'énergie, permettra à long terme de réaliser des économies et de préserver l'environnement.

Par conséquent, c'est au regard de ces différents enjeux que le présent ouvrage se concentre sur le développement de l'une de ces méthodes. En effet, il s'agira d'élaborer un modèle de prévision de la consommation d'énergie à l'aide d'un Réseau Bayésien Dynamique (RBD). Cette méthode, qui se base sur une approche probabiliste plutôt que fréquentiste, se veut prometteuse dans la mesure où elle permettrait de dépasser le problème de « boîte noire » rencontré par les modèles d'apprentissage automatique et profond.

Ainsi, c'est dans le but de développer ce modèle prédictif que lesdits travaux ont été structurés comme suit.

Dans un premier temps, il sera question d'effectuer une synthèse bibliographique de l'ensemble des méthodes d'analyse de données employées dans le cadre de la prévision de la consommation d'énergie. On retrouvera dans cette première partie les objectifs des études, les méthodologies employées et les résultats des modèles.

La deuxième partie, quant à elle, s'articulera autour de la présentation de la base de données UK-DALE. Elle comprendra une description des techniques de collecte de données impliquant l'utilisation des *smart meters* et un inventaire de la base de données (identifiants, variables, etc.).

Enfin, la dernière partie portera sur le déploiement de l'algorithme de prévision. Cette partie détaillera la méthodologie retenue dans la construction du modèle de prévision, les analyses exploratoires, les résultats du Réseau Bayésien et les perspectives des futurs travaux.

1 REVUE DE LITTERATURE

La prévision de la consommation d'énergie est un sujet complexe, qui comporte notamment des enjeux économiques et climatiques. Cette thématique suscite un vif intérêt au sein de la communauté scientifique qui ne cesse de proposer des modèles de plus en plus sophistiqués afin d'atteindre l'optimalité en matière d'efficience énergétique. Cette synthèse bibliographique présente un aperçu non exhaustif des principales méthodes d'analyse de données utilisées à ce jour.

1.1 MODELES CONVENTIONNELS DE REGRESSION ET DE SERIES TEMPORELLES

Les modèles conventionnels regroupent les modèles de régression linéaire, non linéaire, ainsi que ceux basés sur les séries temporelles. Ces approches font partie des premières méthodes d'analyse de données employées dans le cadre de cette thématique. Elles ont notamment été utilisées pour prédire la consommation d'énergie à l'échelle nationale, sectorielle et résidentielle.

En 1994, les premiers modèles de prévision de la consommation d'énergie apparaissaient déjà dans l'étude d'Al-Garni (1994). En effet, dans leurs travaux, Al-Garni *et al.* ont proposé un modèle de prévision de la consommation d'électricité basé sur une régression linéaire multiple, à partir de données collectées entre 1987 et 1992 en Arabie Saoudite. Leur modèle prenait pour variable à expliquer la consommation mensuelle d'électricité et pour variables explicatives : la température ambiante, la taille de la population, l'humidité, et les radiations solaires.

Plus tard, en 2006, Murat *et al.* ont poursuivi dans cette voie, en utilisant cette fois la consommation nette d'électricité comme variable cible. Les variables explicatives, quant à elles, se rapportaient au coût de l'électricité de quatre catégories de source d'énergie.

Cependant, ces méthodes de régression linéaire ont progressivement été délaissées au profit des modèles de régression non linéaire, jugés plus adaptés pour modéliser le caractère complexe et non linéaire de la consommation d'énergie.

C'est dans ce sens que s'inscrivent les travaux de Hao *et al.* (2015), qui ont développé trois modèles de régression intégrant une transformation logarithmique afin de prédire la consommation de charbon en 2020. Le présent ouvrage n'approfondit pas davantage cette catégorie de méthodes, dans la mesure où plusieurs études bibliographiques, telles que celles de Azhar *et al.* (2016), en proposent déjà une analyse plus détaillée.

Bien que les modèles de régression non linéaire aient permis une avancée significative dans la prévision de la consommation d'énergie, ils présentent, tout comme les modèles linéaires, une limite structurelle d'ordre mathématique : ils ne prennent pas en compte le caractère temporel du phénomène associé à la consommation d'énergie. En résumé, les modèles de régression ne parviennent pas à capturer la dépendance historique entre les observations.

Ainsi, afin de surmonter cette limitation, les modèles de séries temporelles ont été largement développés. En effet, grâce à l'intégration de notions telles que l'autocorrélation et l'autocovariance, ces modèles permettent de prendre en compte l'historicité des données de consommation d'énergie dans l'élaboration de modèles prédictifs. Très efficaces, ils ont été utilisés dans de nombreuses études.

Parmi elles, on peut citer celle de Chujai et Kerdprasop (2013), qui propose une comparaison entre les modèles ARMA et ARIMA pour la prévision de la consommation d'électricité à court, moyen et long termes. Les modèles ont été estimés à l'aide de la méthodologie de

Box-Jenkins, et les résultats ont montré que le modèle ARIMA est préférable pour les prévisions mensuelles et semestrielles, tandis que le modèle ARMA convient mieux aux périodicités journalières et hebdomadaires. Les critères d'évaluation utilisés dans cette étude sont l'AIC et le RMSE.

En 2014, Yasmeen et Sharif ont développé et comparé quatre modèles de séries temporelles – ARIMA, SARIMA, ARCH et GARCH – pour la prévision de la consommation mensuelle d'électricité. Ces modèles, entraînés à partir de données collectées en Palestine entre janvier 1990 et décembre 2011, ont été estimés et évalués à l'aide de trois critères : l'AIC, le BIC et le MAPE. Les deux premiers ont permis de déterminer que les modèles les plus appropriés sont ARIMA(3,1,2), SARIMA(2,1,2), ARCH(2) et GARCH(1,1). Le MAPE, utilisé pour évaluer les performances prédictives, a révélé que le modèle ARIMA est le plus précis, avec un MAPE de 5,99, contre 10,51 pour SARIMA, 11,71 pour ARCH et 9,99 pour GARCH.

Enfin, les revues de littérature proposées par Mat Daut *et al.* (2017) et Deb *et al.* (2017) offrent des compléments utiles pour approfondir ces travaux.

1.2 MODELES D'APPRENTISSAGE AUTOMATIQUE

Les hypothèses de normalité des résidus, de non-stationnarité, ainsi que la difficulté liée au caractère non linéaire des données constituent autant de faiblesses des modèles statistiques étudiés précédemment (Mat Daut *et al.*, 2017). Ces limites, combinées à l'émergence des données massives de consommation d'énergie, ont conduit les chercheurs à s'orienter vers des méthodes considérées comme plus modernes, telles que les approches d'apprentissage automatique.

Comme l'illustre la célèbre étude de Dong *et al.* (2005), certains modèles d'apprentissage automatique, notamment les machines à vecteurs de support (SVM), se distinguent par leur grande précision. Cela s'explique en partie par leur capacité à limiter les effets du surapprentissage et à modéliser des phénomènes non linéaires (Mat Daut *et al.*, 2017). Dans cette étude, le SVM, entraîné à partir d'observations mensuelles provenant de quatre bâtiments commerciaux, a obtenu un coefficient de variation (CV) inférieur à 3 % et une erreur de pourcentage inférieure à 4 %.

Compte tenu de cette efficacité, de nombreux autres travaux ont exploré l'usage de l'apprentissage automatique, notamment les méthodes dites *ensemblistes*.

En 2016, Wang et Srinivasan ont comparé les performances d'un arbre de régression (RT) à celles d'une forêt d'arbres décisionnels (EBT) pour la prévision de la consommation d'électricité. Les modèles, intégrant comme variables exogènes les données météorologiques et l'occupation des logements, ont été entraînés sur 89 % des observations, puis testés sur les 11 % restantes. Les résultats montrent que la forêt d'arbres décisionnels présente de meilleures performances que l'arbre de régression, comme en témoignent les valeurs du R^2 (0,90 contre 0,82), du MAPE (3,17 % contre 3,89 %) et du RMSE (0,88 contre 0,93).

À la lumière de ces résultats prometteurs, des revues de littérature plus approfondies ont été menées (Ahmad *et al.*, 2014 ; Deb *et al.*, 2017).

1.3 MODELES D'APPRENTISSAGE PROFOND

L'efficacité des algorithmes d'apprentissage automatique est aujourd'hui indéniable. Cependant, ces derniers nécessitent des optimisations intensives et un temps de calcul relativement long (Ahmad et al., 2014, Amalou et al., 2022). Face à ces contraintes, d'autres méthodes d'apprentissage basées sur des architectures dites « profondes » ont été investiguées. Ils d'agit

des modèles d'apprentissage profond à l'instar des Perceptrons Multicouches (MLP), des Réseaux de neurones Convolutifs (CNN) et des Réseaux de Neurones Récurrents (RNN).

Ainsi, dans leur étude, (Azadeh et al., 2008) ont développé un MLP pour prédire la consommation annuelle d'électricité dans les secteurs à forte intensité énergétique. Le réseau de neurone, caractérisé par une fonction d'activation sigmoïde a obtenu un MAPE de 0.0099 qui est largement inférieur à celui d'un modèle de régression non linéaire évalué à 0.075.

Enfin, (Amalou et al., 2022) ont conçu et comparé un modèle d'apprentissage profond et deux autres modèles hybrides. Il est question d'un réseau de neurone récurrent classique (RNN), d'un modèle de Long Short Term Memory (LSTM) et d'un modèle à Unité récurrente fermée (GRU). Les résultats indiquent que le GRU performe mieux que le LSTM et largement mieux que le RNN classique.

Bien que l'utilisation des réseaux de neurones pour la prévision de la consommation d'énergie soit un sujet de recherche largement documenté (Ahmad *et al*., 2014 ; Deb *et al*., 2017). il est important de noter que ces intelligences artificielles à l'instar des SVM souffrent d'un problème de « boîte noire » qui les prive de toute forme explicabilité (Foucquier et al., 2013)

1.4 MODELES HYBRIDES

Il en va de soi que chacune des méthodes énumérées ci-dessus revêt des avantages comme des inconvénients. De ce fait, c'est dans but de tirer parti des forces uniquement de ces méthodes qu'une approche par hybridation, bien que plus complexes, est de plus en plus privilégiée. Enfin, cette approche qui consiste à fusionner des modèles issues de différentes catégories dans le but dans de conserver uniquement leurs avantages est aujourd'hui considérée comme l'une plus efficace en matière de prévision et d'explicabilité.

Parmi ces modèles, on retrouve le modèles à machine à vecteurs de support des moindres carrés (LSSVM) qui offre un gain considérable de temps de calcul et une meilleure précision que les SVM. Il a notamment été utilisé dans l'étude de (Kaytez *et al.*, 2015) où il est comparé à une régression multiple et un réseau de neurones récurrent. Les résultats indiquent que les prévisions du LSSVM sont plus précises que celles des deux modèles au regard du MAPE et du RMSE.

Enfin, comme dernier modèle hybride proposé, on retrouve les réseaux bayésiens dynamiques qui ont été récemment utilisés pour prédire la consommation d'électricité (Singh et Yassine, 2018). Cependant, l'application de ces réseaux probabiliste est encore moins répandue bien que très prometteur (Aulia *et al.*, 2014).

En définitive, l'efficience énergétique est un enjeu majeur qui cristallise à la fois l'opinion publique et la recherche scientifique. L'une des approches privilégiée pour atteindre cet objectif reste la prévision de la consommation d'énergie. Bien que cette dernière semble être un phénomène complexe à modéliser en raison notamment de son caractère multifactoriel et non linéaire, les méthodes d'analyse de données de plus en plus sophistiquées sont développées, réduisant ainsi, au fil du temps, les erreurs de prévision et le temps de calcul.

2 TABLEAU DE SYNTHESE

Auteurs	Données	Périodicité	Variable d'intérêt	Variables Explicatives	Méthodes	Résultats
Al-Garni et al 1994	Arabie Saoudite (1987-1993)	Mois	Consommation d'électricité	Température, humidité, radiation solaire, population.	Régression Stepwise	L'influence de la variables associée à l'humidité est négligeable au sein du modèle de régression
Tunc et al 2006	Turquie (1980-2001)	Année	Consommation d'électricité	Coûts de production, coûts fixes et coûts variables lié à la construction	Régression linéaire Multiple	Au regard des prévision, les centrales hydroélectriques et nucléaires sont importantes pour la soutenabilité de la consommation d'énergie.
Hao et al 2015	Chine (1995-2012)	Année	Consommation de charbon par habitant	PIB par habitant, Taux d'urbanisation, Taux d'ouverture, valeur ajoutée de l'industrie secondaire	Régression avec une transformation logarithmique	Les prévisions faîtes par le modèle de régression indiquent une croissance de la consommation de charbon en chine jusqu'en 2020.
Chujai et al 2013	France (2006-2010)	Année, Mois, Semaine, Heure	Consommation d'électricité	-	ARMA, ARIMA	Les Résultats indiquent que le modèle ARMA est plus adapté pour les prévisions à long terme d'une part. D'autre part, le modèle ARIMA est plus efficace pour les prévisions à court terme.

Yasmeen and Sharif, 2014	Palestine (1990-2011)	Mois	Consommation d'électricité	Univarié	ARIMA, GARCH, SARIMA	ARIMA (3,1,2) model is the most appropriate model for forecasting electricity consumption of Pakistan.
Wang and Srinivasan 2016	Etats-Unis (2015.02.01-2015.02.28)	Heure	Consommation d'électricité	weather conditions, day types, and time of day	Ensemble Bagging Tree	Les capacités prédictives du modèle EBT sont supérieures à celles de l'arbre de régression (RT) avec une amélioration du MAPE de 18.5%.
Dong et al 2005	Singapour (1996-2001)	Mois	Consommation d'électricité	Univarié	SVM	Comparé aux autres modèles, le SVM présente l'un des coefficient de variation les plus faibles. Il est inférieur à 3%.
Ahmad et al 2014 (revue de littérature)	-	-	-	Univarié	-	Les méthodes d'apprentissage automatique ont comme principales limites : un besoin intensif d'optimisation et un temps de calcul relativement long
Azdeh et al 2008	Iran (1997-2003)	Année	Consommation d'électricité	Le prix de l'électricité, Le nombre de consommateur par secteur, intensité énergétique, valeur ajoutée par secteur	Réseaux de neurones (Perceptron multicouche)	Les résultats montrent que la moyenne des prévisions effectuées par le modèle de MLP est statistiquement identique à celle issues des données réelles avec un intervalle de confiance à 99%.
Amalou et al 2022	Australie (2010-2014)	Heure	Consommation d'électricité	Univarié	Unité récurrente fermée	Le GRU a obtenu un meilleur RMSE que les modèles LSTM et RNN, 0.034 contre 0.039 et 0.051 respectivement.

Kaytez et al 2015	Turquie (1970-2009)	Année	Consommation net d'électricité	La capacité énergétique, la production brute d'électricité, La population, Le total des abonnements	LS-SVM	Le modèle LSSVM a obtenu des meilleurs résultats que les modèles MLR et ANN de 1,70 % et 0,88 % respectivement.
Aulia et al, 2024 (revue de littérature)	-	-	-		-	Les réseaux bayésiens dynamiques surpassent les modèles d'apprentissage automatique et profond en termes d'
Zhang et al 2024	Chine (2020-2021)	Minute	Production d'énergie solaire	weather information, operation indicators, sensor data, and output AC power	Réseau Bayésien Dynamique	Le réseau bayésien présente une meilleure gestion de l'incertitude.
Singh et al 2018	Royaume-Unis (2012-2015) Canada (2012-2014)	Minute	Consommation d'électricité	hour of the day, period of the day, weekday, week, month and season of the year as well as appliance-appliance associations	Réseau Bayésien Dynamique	Le modèle de RBD présente des Résultats supérieurs à ceux du SVM et du MLP.

Tableau 1: Tableau récapitulatif de l'ensemble des articles académique consultés

3 ETUDE EMPIRIQUE

3.1 DONNEES

3.1.1 Présentation du programme de recherche UKDALE

Au regard de la nécessité de concevoir des algorithmes de plus en plus sophistiqués, plusieurs programmes de recherche ont été mis en place dans le but de faciliter l'accès aux données de consommation d'énergie. Parmi ces initiatives, on retrouve la base de données UK-DALE (UK – Domestic Appliance-Level Electricity), développée par le département d'informatique de l'Imperial College London (Kelly & Knottenbelt, 2015).

Disponible en open source, UK-DALE est aujourd'hui considérée comme une référence parmi les bases de données de consommation d'énergie en raison de la richesse et de la précision des informations qu'elle fournit. Elle contient notamment des données de panel à haute fréquence : 16 kHz pour la consommation globale au niveau du foyer et 1 Hz ou 6 Hz pour la consommation par appareil. Ces données ont été collectées entre 2012 et 2015 auprès de cinq foyers, à l'aide de compteurs intelligents (« smart meters »).

Les compteurs intelligents sont des dispositifs électroniques conçus pour mesurer la consommation d'énergie en temps réel. Leur utilisation est de plus en plus répandue, notamment dans le contexte actuel marqué par la transition énergétique, l'essor de l'intelligence artificielle, et l'intégration croissante des nouvelles technologies de l'information et de la communication (NTIC) dans la vie quotidienne (smart city, smart home, smartphone).

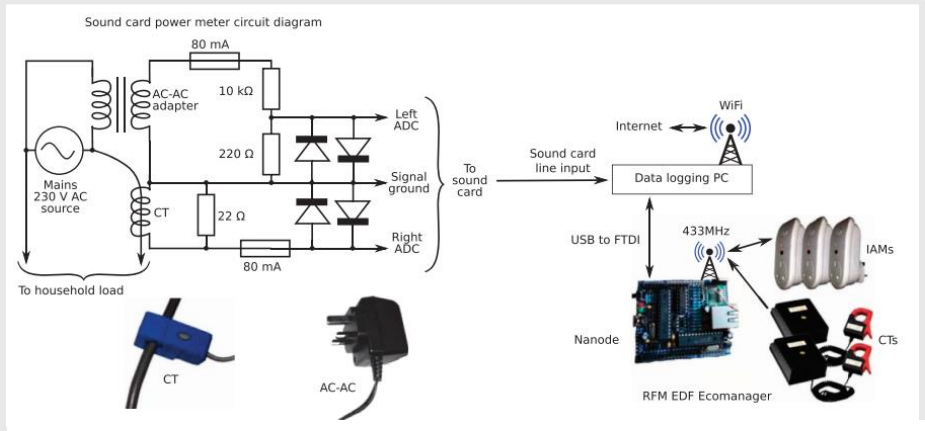

Figure 1: Système de collecte de données. Extrait (Kelly1 & Knottenbelt1, 2015)

Au sein de UK-DALE, ces capteurs ont notamment permis de mesurer, collecter et stocker les données de consommation d'énergie qui sont organisées en dossiers, sous-dossiers et fichiers dans lesquels on retrouve :

- Les données de consommations d'électricité (puissance active, puissance apparente, tension...)
- Les données par type de bâtiments (Nombre d'occupants, année de construction, ...)
- Les données de contrôle (date d'enregistrement, date d'installation des capteurs, date de la première mesure, date de la dernière mesure, ...)
- Les types d'appareils par résidence (réfrigérateur, Télévision, Lave-vaisselles, Micro-ondes, ...).

Toutes ces informations ont été résumées dans le tableau ci-dessous.

Foyer	1	2	3	4	5
Type de bâtiment	maison de bout de terrasse	maison de bout de terrasse		Maison en milieu de rangée	Immeuble résidentiele
Année de construction	1905	1900		1935	2009
Système de chauffage	gaz naturel	gaz naturel		gaz naturel	gaz naturel
Nombre d'occupants	4	2		2	2
Nombre de dispositifs de mesure par appareil	54	20	5	6	26
Nombre de dispositifs de mesure principal	2	2	1	1	2
Date de la première mesure	2012-11-09	2013-02-17	2013-02-27	2013-03-09	2014-06-29
Date de la dernière mesure	2015-01-05	2013-10-10	2013-04-08	2013-10-01	2014-11-13
Durée d'enregistrement (jour)	786	234	39	205	137
Durée d'enregistrement des mesures	655	140	36	155	131
Consommation moyenne d'électricité par jour (puissance active kWh)	7.64	7.17			13.75
Consommation moyenne d'électricité par jour (puissance apparente kVAh)	8.90	8.00	12.35	10.24	17.56

Tableau 2: Données relatives à chaque Foyer. Extrait (Kelly1 & Knottenbelt1, 2015)

3.1.2 Préparation de la base de données

Les données massives de UK-DALE, disponibles au format HDF5, ont été exploitées à l'aide de divers outils tels que Python, Microsoft Azure, Databricks et SQL.

Tout d'abord, Python a été utilisé pour lire les fichiers, identifier l'architecture de la base de données (dont un aperçu est présenté ci-après) et convertir les fichiers au format CSV.

Ensuite, le langage SQL a été employé dans l'environnement cloud Microsoft Azure, via la plateforme Databricks, pour manipuler les données brutes. Cette étape a permis notamment le changement de type de certaines variables, la création de nouvelles variables, ainsi que l'inventaire des bases de données.

Toutes ces procédures ont été indispensables pour obtenir des données de consommations d'énergie de bonne qualité. Pour illustrer ce travail, un aperçu d'une table de données avant et après manipulation est présentés (ci-dessous).

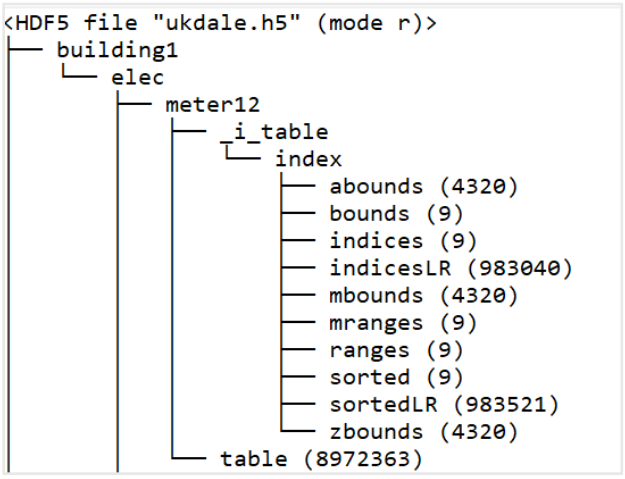

Figure 2 :Extrait de l'architecture des données de consommation du [réfrigérateur] du Foyer 1

Dans ces tables, on peut distinguer principalement 2 catégories de variable. La première catégorie indique la date à laquelle les données ont été enregistrées. Cette dernière englobe la variable unix et toutes les nouvelles variables temporelles présentent dans la base de données traitées (unix_year, unix_monthy, unix_day, unix_hour, unix_minute, unix_second).

La deuxième catégorie quant à elle représente les mesures associées à la consommation d'électricité. On y retrouve la puissance active (active_power) exprimée en Watt (W), la puissance apparente (apparent_power) exprimée en Volt-ampère (VAh) et la tension électrique (main_rms_voltage) exprimée en Volt (V).

unix	active_power	apparent_power	main_rms_voltage
1.363548e+09	337.88	431.04	240.15
1.363548e+09	339.43	427.94	240.56
1.363548e+09	340.63	429.66	241.07
1.363548e+09	338.80	426.99	240.44
1.363548e+09	340.88	429.13	241.01

Tableau 3: Extrait de données brutes de consommation d'énergie du Foyer 1

unix	unix_year	unix_month	unix_day	unix_hour	unix_minute	unix_second	active_power	apparent_power	main_rms_voltage
2013-03-17 19:12:43.099999905	2013	3	17	19	12	43	337.88	431.04	240.15
2013-03-17 19:12:44.099999905	2013	3	17	19	12	44	339.43	427.94	240.56
2013-03-17 19:12:45.099999905	2013	3	17	19	12	45	340.63	429.66	241.07
2013-03-17 19:12:46.099999905	2013	3	17	19	12	46	338.80	426.99	240.44
2013-03-17 19:12:47.099999905	2013	3	17	19	12	47	340.88	429.13	241.01

Tableau 4: Extrait de données traitées de consommation d'énergie : Foyer 1

3.1.3 Détermination de la variable dépendante

Après la préparation des bases de données, l'étape suivante consiste à choisir la variable indépendante qui est associée à la consommation d'énergie. Toutefois, il paraît important de rappeler que l'énergie peut revêtir plusieurs formes à l'instar de l'électricité, du gaz et de l'eau, etc. De ce fait, la consommation d'énergie se présente alors comme un phénomène polymorphe à modéliser. Néanmoins, comme dans beaucoup d'autres études, le présent ouvrage substitue la consommation finale d'énergie à la consommation finale d'électricité par souci de simplification. Ce choix repose sur le principe selon lequel l'électricité occupe la plus grande part dans la consommation finale d'énergie après le pétrole. En d'autres termes, il s'agira de modéliser la consommation d'énergie par la consommation d'électricité en utilisant comme variable dépendante la puissance active exprimée en Watt.

3.2 ANALYSES EXPLORATOIRES

3.2.1 Statistiques descriptives

Critères	Variables	Foyer 1	Foyer 2	Foyer 5
Max	Puissance active	2829.45	1863.51	3530.69
	Puissance apparente	2902.86	1877.75	3626.27
	Tension	246.64	244.63	250.59
Min	Puissance active	87.51	0.00	226.11
	Puissance apparente	125.26	145.60	399.48
	Tension	237.77	236.62	238.76
Moyenne	Puissance active	318.55	298.63	571.98
	Puissance apparente	371.27	333.24	730.42
	Tension	243.18	241.15	246.12
Médiane	Puissance active	222.43	207.76	429.95
	Puissance apparente	264.29	240.36	601.73
	Tension	243.23	241.19	246.10
Q1	Puissance active	155.27	143.33	316.88
	Puissance apparente	196.04	174.12	488.35
	Tension	242.41	240.42	244.96
Q3	Puissance active	408.86	337.10	625.90
	Puissance apparente	461.63	370.68	780.57
	Tension	244.00	241.94	247.34
Ecart-type	Puissance active	242.06	260.33	396.12
	Puissance apparente	261.11	261.29	377.18
	Tension	1.17	1.13	1.68

Variance	Puissance active	58591.53	67772.01	156913.65
	Puissance apparente	68179.43	68272.85	142265.37
	Tension	1.36	1.27	2.81
CV (%)	Puissance active	75.99	87.18	69.26
	Puissance apparente	70.33	78.41	51.64
	Tension	0.48	0.47	0.68
Skewness	Puissance active	2.36	2.77	2.52
	Puissance apparente	2.37	2.70	2.55
	Tension	-0.29	-0.20	-0.12
Kurtosis	Puissance active	8.40	8.84	8.04
	Puissance apparente	7.96	8.32	8.43
	Tension	0.07	0.13	-0.17
Valeurs nulles	Puissance active	0.00	0.00	0.00
	Puissance apparente	0.00	0.00	0.00
	Tension	0.00	0.00	0.00

Tableau 5 : Tableau de Statistique de la consommation d'énergie (kWh) en heure (H).

L'analyse exploratoire est une étape essentielle dans le développement des algorithmes de prévision. Elle permet d'extraire les principales caractéristiques de l'évolution de la consommation d'énergie en fonction des individus qui sont en l'occurrence les foyers et des périodicités qui peuvent être annuelles, mensuelles, journalières, horaires, en minute ou en seconde.
Tout au long de cette étude, seuls trois individus parmi les cinq énoncés plus haut ont été retenus. Par ailleurs, l'heure (H) a été sélectionnée comme périodicité de référence. Ces précisions visent à encadrer l'étude en facilitant l'interprétation des résultats, la comparaison des modèles prédictifs et la reproduction de ces travaux.

Les paramètres de tendance centrale révèlent que le foyer 5 a une consommation horaire supérieure aux foyers 1 et 2 en moyenne et en médiane. En effet, la consommation horaire d'électricité du foyer 5 s'élève environ à 572W contre 319W et 299W qui correspondent respectivement aux foyers 1 et 2.

Dans le même ordre, il a été enregistré dans le foyer 5 un pic de consommation horaire d'électricité avoisinant les 3531W contre 2830W et 1864W qui correspondent aux foyers 1 et

2. Ces chiffres sont à mettre en opposition avec la consommation minimale d'électricité dont la plus petite mesure appartient au foyer 2 qui s'élève à 0W. Cela suggère que durant toute la période d'observation, au moins un appareil électrique est resté actif au sein des foyers 1 et 2. Ceci diffère du foyer 2, qui laisse entrevoir des périodes sans aucune mesure de consommation électrique.

S'agissant maintenant des paramètres de dispersion, on constate aisément que c'est au sein du foyer 5 que l'on mesure la plus grande variabilité en matière de consommation horaire d'électricité. En effet, l'écart-type de ce dernier s'élève à 396.12W contre 242.06W et 260.33W. Ces derniers chiffres se réfèrent respectivement aux foyers 1 et 2.

En définitive, en se basant sur ces paramètres, on peut conclure que le foyer 5 consomme en moyenne plus d'électricité que les foyers 1 et 2 par heure, d'une part. D'autre part, cette consommation est beaucoup plus volatile. De plus, on ne dénombre aucune valeur manquante. Ce type de constat est très caractéristique des bases de données cylindrées à l'instar de UKDALE qui ont une forte sélection en matière d'échantillon.

3.2.2 Le diagramme de Tukey

La consommation de l'électricité est un phénomène temporel marqué fortement par un caractère saisonnier (cyclique). Comme les graphiques ci-dessous le suggèrent, ce phénomène revêt une courbe en forme de double cloche. Cette dernière a tendance à diminuer durant les heures de coucher (21h-5h) avant de fortement augmenter durant les heures d'activité (6h-9h) et (17h-19h).

Aussi, tout au long de la journée, la variance de celle-ci augmente avant d'atteindre un pic de consommation dans les alentours de (19h-20h). Ces différents comportements semblent être structurels.

Pour résumer, ce graphique a permis de mettre en évidence certaines caractéristiques de la consommation d'électricité à l'instar de la saisonnalité dans le cadre d'une périodicité par heure.

Les analyses effectuées sur cet échantillon semblent correspondrent aux observations réelles.

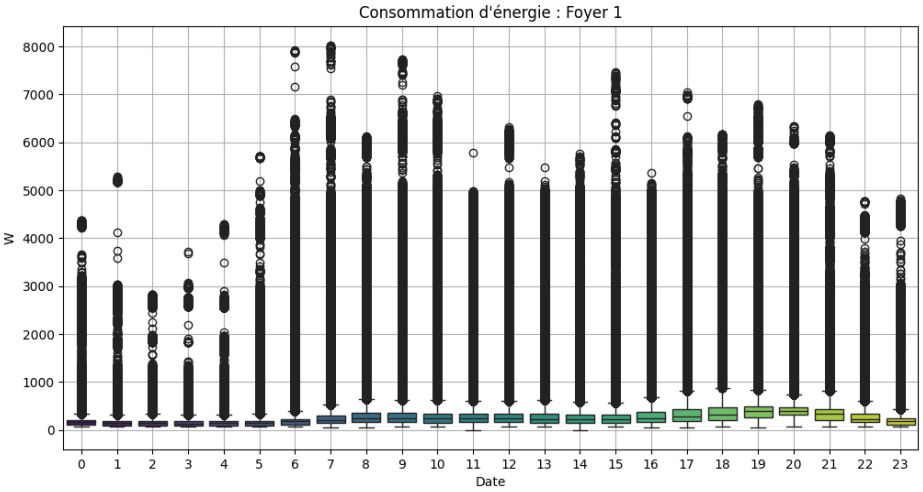

Figure 3 : Diagramme de Tukey representation la consommation d'électricité du foyer 1 par heure

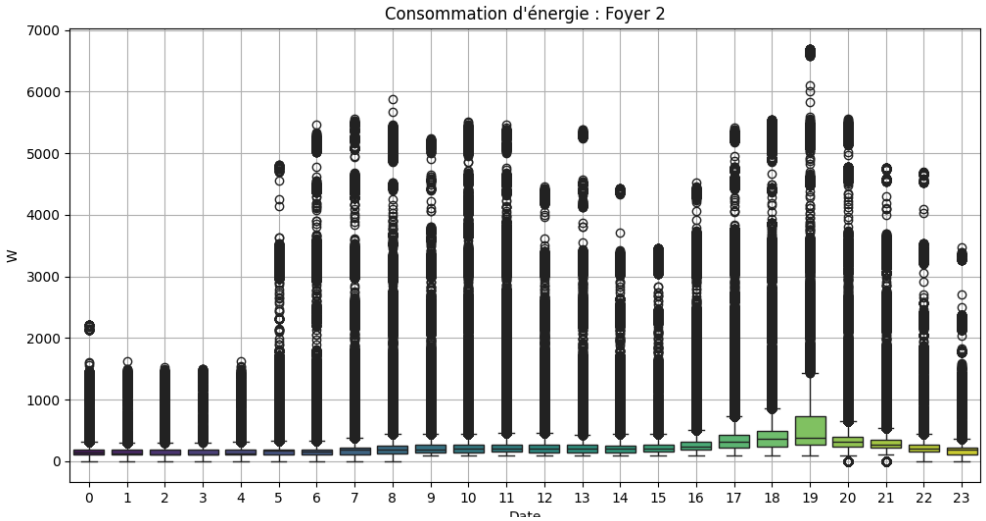

Figure 4 : Diagramme de Tukey representation la consommation d'électricité du foyer 2 par heure

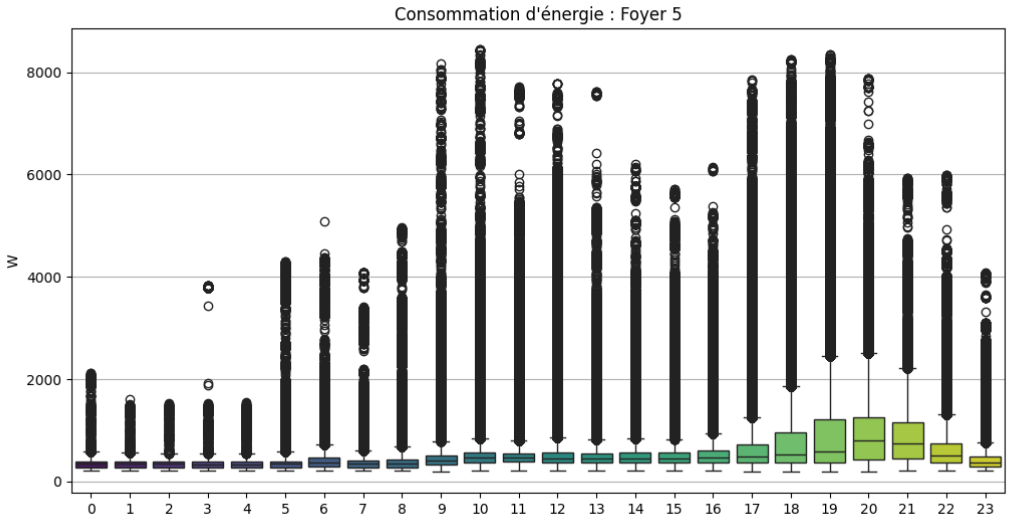

Figure 5 : Diagramme de Tukey representation la consommation d'électricité du foyer 2 par heure

3.2.3 Séries chronologiques

Représenter visuellement la consommation d'électricité sous forme d'une série chronologique comporte plusieurs avantages tels que la possibilité d'observer des tendances, d'identifier des comportements périodiques et de détecter des anomalies.

Ainsi, tout comme les diagrammes de Tukey, l'analyse visuelle permet d'identifier la présence d'un comportement saisonnier. Bien que ce dernier ne soit plus visible en heures, on parvient à l'identifier maintenant en mois. Ainsi, on visualise aisément une hausse potentiellement structurelle de la consommation d'électricité dans la période d'octobre à janvier et une baisse de cette dernière entre les mois d'avril et d'août.

Ces variations de la consommation d'électricité semblent lier au climat et plus précisément au changement des saisons au Royaume-Uni. C'est en hiver, qu'on observerait des pics récurrents de la consommation d'électricité et c'est en automne que celles-ci seraient les plus basse.

Dans ces graphiques, l'information la plus importante semble être la continuité de la série. Les séries chronologiques des foyers 2 et 5 sont affectées par des niveaux de consommation d'électricité anormaux. Par exemple, on observe, au sein du foyer 2, entre fin août 2013 et fin septembre 2013, une augmentation linéaire du niveau de la puissance.

Concernant le foyer 5, cette anomalie est moins prononcée car elle dure moins d'un mois et à la différence de celle du foyer 2, elle présente une tendance linéaire vers le bas. Il est important de noter qu'aucune information n'a été donnée à ce sujet. L'une des hypothèses les plus probables serait des défauts d'enregistrement de la part des smart meters.

En définitive, toutes ces analyses ont servi à illustrer la complexité des données relatives à l'évolution de la consommation d'électricité dans le temps. Cette complexité est marquée, entre autres, par un caractère saisonnier qui dépend de la périodicité de référence, des anomalies qui peuvent être imputées aux erreurs de mesure, aux comportements des personnes au sein des foyers et à la présence de facteurs exogènes comme le climat et le nombre d'habitants.

Toutes ces contraintes doivent être prises en compte dans la construction des modèles de prévisions pour garantir des résultats fiables et de bonnes performances prédictives.

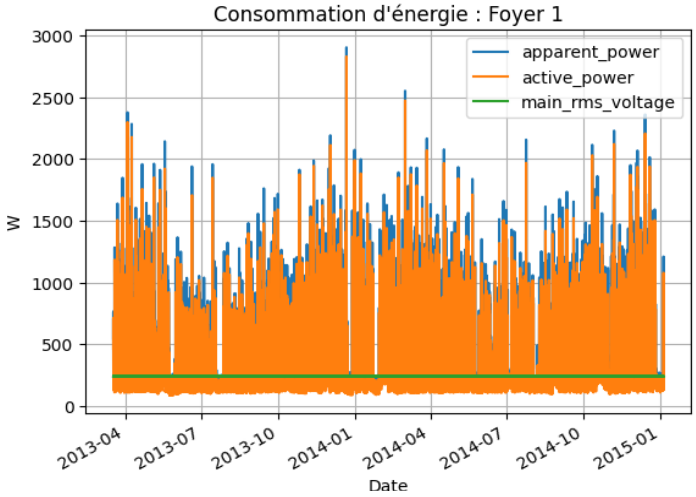

Figure 6 : Série chronologique de la consommation d'électricité du Foyer 1 par heure

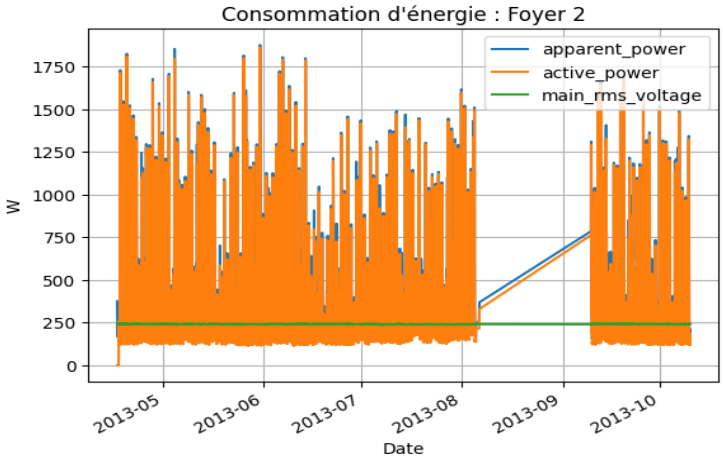

Figure 7 : Série chronologique de la consommation d'électricité du Foyer 2 par heure

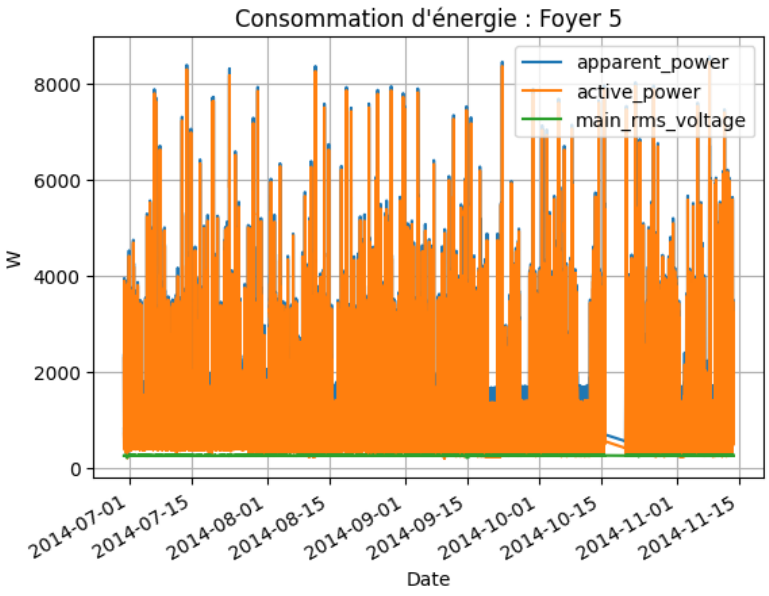

Figure 8 : Série chronologique de la consommation d'électricité du Foyer 5 par heure

4 MODELES DE PREVISION

Dans cette étude, deux modèles de prévision de la consommation d'électricité ont été développés. Il s'agit d'un modèle de séries temporelles ARIMA et d'un Réseaux Bayésien Dynamique. L'objectif est de comparer les performances du modèle graphique probabiliste à celles d'un modèle conventionnel d'analyse de données.

4.1 MODÈLE ARIMA

Le modèle AutoRegressive Integrated Moving Average abrégé ARIMA est un modèle de séries temporelle univarié qui peut-être définit comme une généralisation du modèle ARMA. Il est composé de trois parties. La première partie appelée Autorégressive (AR) est une série chronologique dans laquelle les valeurs présentes sont modélisées par ses valeurs passées. La deuxième partie constitue la partie Moyenne Mobile (MA). Cette partie vise à modéliser la série chronologique par ses termes d'erreur passés. La combinaison de ces deux parties (AR) et (MA) forme un processus ARMA. Cependant, l'usage de ce dernier suppose une condition de stationnarité qui n'est pas toujours vérifiée.

Par conséquent, c'est dans le but de rendre le processus ARMA stationnaire qu'une troisième composante appelée Integrated représentée sous la forme d'un (I) est ajoutée. Cette dernière permet de rendre la série temporelle stationnaire en procédant à des calculs de différenciation.

Une fois la série différenciée et les hypothèses du modèle vérifiées, les paramètres du modèle peuvent être estimés et utilisés pour effectuer des prévisions.

Ainsi, c'est dans le but de réaliser des prévisions de la consommation d'électricité que le modèle ARIMA a été sollicité. Ce dernier a été développé en utilisant la méthodologie de Box-Jenkins qui se présente comme suit :

4.1.1 Etude de la stationnarité

La première étape consiste à identifier si le phénomène associé à la consommation d'électricité au cours du temps est un processus stationnaire. Pour se faire, plusieurs méthodes ont été déployées. Parmi elles, on retrouve une analyse des autocorrélations (AFC) et des autocorrélations partielles (PAFC), une décomposition de la consommation d'électricité en tendance, saisonnalité et résidus et enfin des tests de Dickey-Fuller.

- L'analyse des autocorrélations et des autocorrélations partielles révèle une stationnarité faible. En effet, les autocorrélations décroient progressivement avant de remonter tout légèrement après le 18ème retard. Quant aux autocorrélations partielles, elles chutent brusquement le plus souvent aux alentours du premier ordre avant de remonter légèrement vers le 18ème retard.

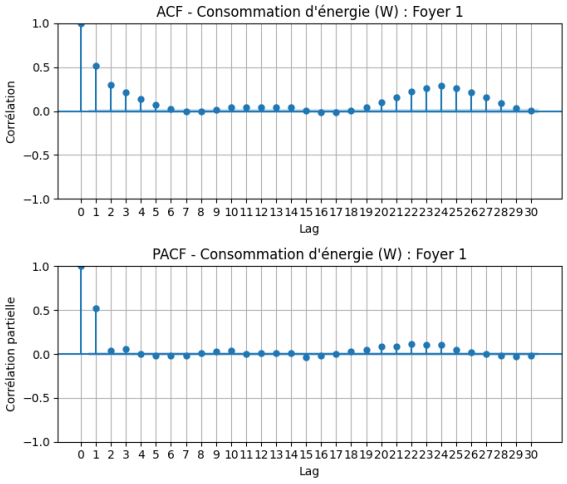

Figure 11 : : ACF et PACF : foyer 1

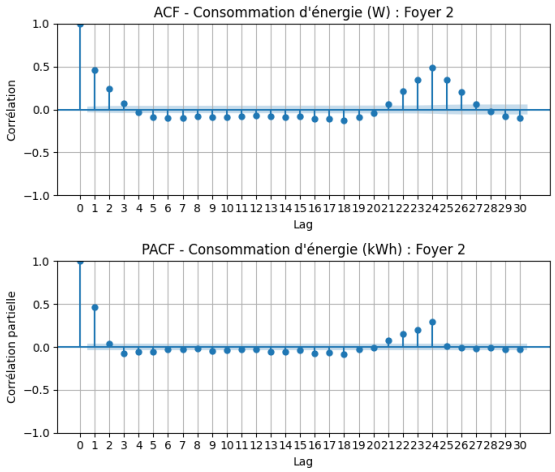

Figure 10 : : ACF et PACF : foyer 2

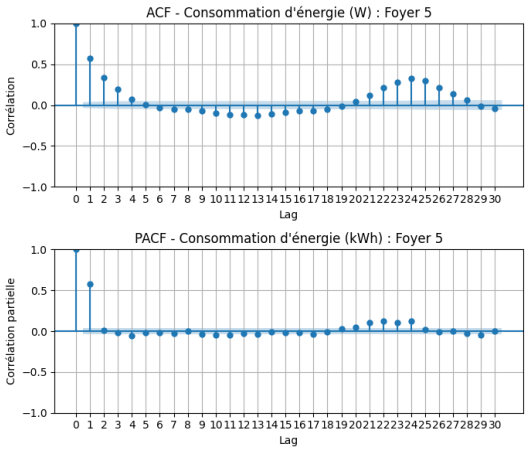

Figure 9 : : ACF et PACF : foyer 5

- La décomposition de la série laisse transparaître une tendance légèrement descendante sur le long terme. Les composants saisonniers semblent bruités. Enfin, les résidus sont assez hétérogènes. On ne parvient pas à identifier des structures évidentes au sein des données.

- Au regard des pvaleurs des tests de Dickey-Fuller, on peut affirmer que la puissance active est un processus stationnaire même si des légères tendances ont été observées.

	Foyer 1	Foyer 2	Foyer 5
Valeur de test	-12.85	-7.88	-7.92
Pvaleur	5.49e-24	4.57-12	3.57-12
Conclusion	Stationnaire	Stationnaire	Stationnaire

Tableau 6 : Tableau récapitulatif du test de Dickey-Fuller

4.1.2 La différenciation

Bien que le processus peut sembler faiblement stationnaire au regard des éléments énoncés ci-dessus, des précautions ont tout de même été prises pour différencier dans certains cas la série. L'objectif étant d'obtenir les meilleurs estimations pour effectuer les meilleures prévisions, mêmes les cas de séries différenciées ont été étudiés. Cette étape a fait intervenir le graphique de différenciation.

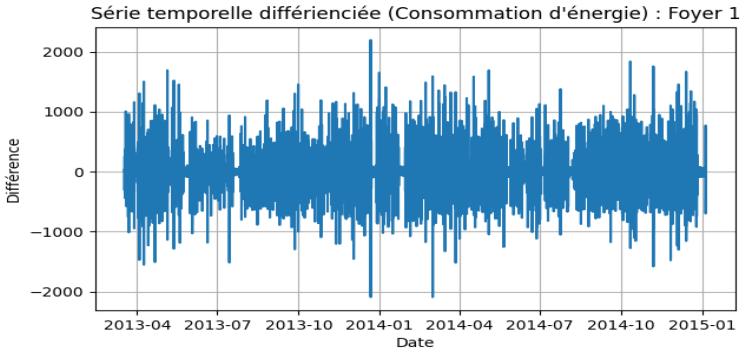

Figure 12 : Série différenciée : Foyer 1

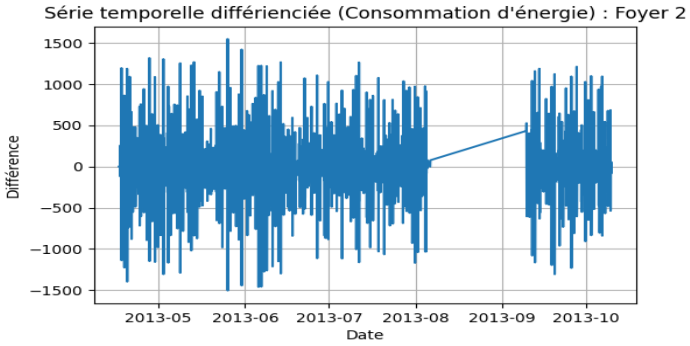

Figure 13 : Série différenciée : Foyer 2

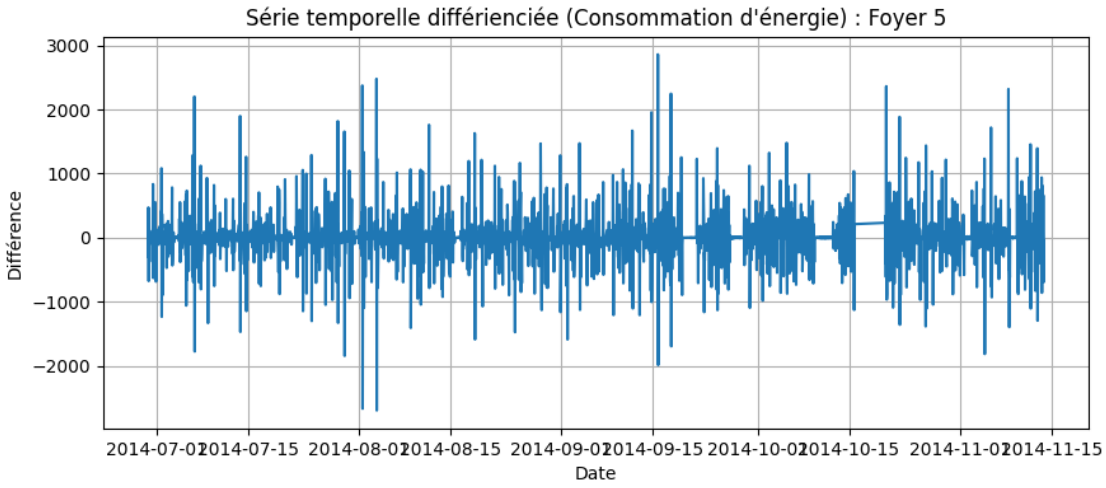

Figure 14 : Série différenciée : Foyer 5

4.1.3 Identification des modèles ARIMA

L'identification des modèles ARIMA s'est faite de façon itérative jusqu'à trouver les ordres qui minimisaient la perte d'information. Ces informations ont été synthétisées dans le tableau ci-dessous avec les paramètres estimés des modèles.

4.1.4 Estimation des paramètres du modèles

	Foyer 1	Foyer 2	Foyer 5
Model :	(1, 1, 1)	(5, 0, 3)	(0, 1, 1)
No Observations	12596	2706	2539
HBIQ	168985.724	37153.268	36839.115
Ljun-Box (Q)	0.45	0.01	9.26
Prob (Q)	0.50	0.91	0.00
Hétéroscédasticité (H)	1.18	0.75	0.98
Prob (H)	0.00	0.00	0.76
Jarque-Bera (JB)	103769.31	15696.62	11679.90
(JB)	0.00	0.00	0.00

Tableau 7 : Tableau récapitulatif des paramètres des Modèles ARIMA

Les modèles ARIMA des foyers 1, 2 et 5 ont été estimés à partir des échantillons de tailles respectives 125596, 27006 et 2539 observations. Ces observations sont en l'occurrence les mesures horaires de la consommation d'énergie durant 80% de la période d'enregistrement des données. Ces chiffres garantissent une fiabilité des résultats.

Le test de Ljung-Box est un test statistiques qui est utilisé pour vérifier si les résidus d'une série temporelle présentent une autocorrélation, c'est-à-dire une dépendance entre les observations t et t-1. C'est une hypothèse nécessaire pour obtenir des résidus aléatoires (bruits blancs) au sein d'un modèle ARIMA. Comme on peut le souligner les pvaleurs des foyers 1 et 2 sont inférieures à 5% donc les résidus de ces modèles sont des bruits blancs.

Le test d'Hétéroscédasticité sert quant à lui à vérifier si la variance d'une série temporelle est constante ou si elle dépend du temps. Cette hypothèse intervient dans la vérification de la stationnarité du processus. D'après le tableau, on peut conclure la présence d'une hétéroscédasticité au sein de processus de consommation d'électricité associé aux foyer 1 et 2. Néanmoins, on observe que la pvaleur du test d'hétéroscédasticité est supérieure à 5% donc, la variance de la consommation d'électricité est stable au sein du foyer 5.

Enfin, le test de Jarque-Bera vise à s'assurer que les résidus du modèle sont distribués selon une loi normale. Comme tous les foyers une pvaleur inférieure à 5%, on peut conclure que les résidus sont non normaux.

4.1.5 Résultats et Discussion

Dans cette étude, une méthode de validation du modèle a été utilisée. Cette méthode consiste à séparer les jeux de données en deux parties. La première partie est constituée à partir 80% du jeu de données brute. Cette partie appelée base de données d'entraînement sert à entraîner les modèles de séries temporelles ARIMA. La deuxième partie quant à elle représente les 20% des données restantes. Elle est constituée à partir de la première observation après la dernière observation de la base de données d'entrainement jusqu'à la fin de la base de données.

A la suite de cela, une évaluation sera effectuée entre les valeurs prédites du modèles et les données réelles regroupées dans la base de données test. Ainsi, pour comparer les performances des modèles ARIMA, le choix s'est porté sur les critères de la log-vraisemblance, l'AIC, le BIC, le MAE, le MAPE et le RMSE.

Métriques	Foyer 1	Foyer 2	Foyer 5
Log Likelihood	-84486.127	-18555.961	-18415.439
AIC	168978.253	37131.923	36834.879
BIC	169000.576	37190.955	36846.557
MAE	193.888374	175.349217	306.666331
MAPE	0.594796	0.784164	0.596263
RMSE	276.896095	254.146390	435.986433

Tableau 8 : Performances prédictives des modèles ARIMA

Avant d'interpréter des résultats des modèles, il parait important de rappeler définir les différentes métriques et de souligner leurs différences.

Log-Likelihood, l'AIC, et le BIC sont des critères d'information qui permettent de comparer la qualité des modèles. Plus la log-vraisemblance est élevée meilleur est l'ajustement. Les critères AIC et BIC servent à pénaliser la complexité du modèle. Le but étant d'obtenir des modèles avec les plus efficaces possibles avec le faible nombre de paramètres.

L'Erreur Absolue Moyenne traduis en anglais (MAE) est indicateur qui est utilisé pour évaluer les performances prédictives d'un modèle. Il se base sur la moyenne des erreurs absolues entre les valeurs observées et les valeurs prédites, sans tenir compte du signe de l'erreur. Dans le cadre des analyse de séries temporelles il est très utile pour obtenir une vision globale de la précision des prévisions.

Dans notre jeu de données, c'est le modèle développé à partir des observations du foyer 2 qui présente les meilleurs résultats. En effet, ce dernier, a obtenu un MAE de 175.35 contrairement aux foyers 1et 5 qui ont obtenu respectivement des MAE qui valent 306.67 et 193.89

Au regard maintenant du pourcentage d'erreur moyen (MAPE), c'est le foyer 1 qui a obtenu un meilleur score. Autrement dit, les prévisions du foyer 1 présentent en moyenne des erreurs de l'ordre de 60%.

Enfin, le RMSE, l'erreur quadratique moyenne est la mesure qui est la plus sensible aux grandes erreurs de prédiction. Concernant, cette métrique, c'est le modèle entrainé à partir des données du foyer 2 qui performe le mieux.

En définitive, le modèle ARIMA a obtenu des performances assez faibles et hétérogènes en matière de prévision de la consommation

4.2 Réseau Bayésien Dynamique

Les Réseaux Bayésiens Dynamiques sont des variantes des Réseaux Bayésiens classiques. Un réseau bayésien est un modèle graphique probabiliste qui, à partir de variables aléatoires structurées en un graphe orienté acyclique, permet de calculer des probabilités conditionnelles liées à ces variables. Les réseaux bayésiens dynamiques étendent ce processus en prenant en compte l'évolution des variables aléatoires, généralement dans le temps. Ces graphes probabilistes sont composés :

- Des nœuds qui représentent des variables aléatoires qui peuvent être observables ou latentes.
- Des arcs qui sont associées aux dépendances temporelles d'une variable à l'instant t (intra-slice arcs) ou à l'instant t et t+1 (inter-slice arcs).

Dans cet ouvrage, chaque nœud au sein du réseau est modélisé par une variable aléatoire continue suivant une distribution normale. Il s'agit ainsi d'un Réseau Bayésien Dynamique Gaussien.

4.2.1 Préparation de la base de données

Avant le déploiement du modèle probabiliste, l'une des étape consistait à préparer les données. Cette phase de préparation s'est déroulée en une succession d'opérations qui se présente comme suit : La transformation des variables périodiques discrètes en variables périodiques continues, La suppression de toutes variables discrètes (unix_year, unix_month, unix_day, unix_hour, unix_minute et unix_second), La normalisation des variables associées

à la puissance apparente et à la tension. Et enfin, à division de la base de données brute en base de données d'entrainement (80%) et de base de données test (20%).

4.2.2 Apprentissage du Réseau Bayésien Dynamique

L'apprentissage du réseau bayésien a effectué en utilisant comme le Critère d'information Bayésien (BIC) comme critère de score. Ce critère permet d'obtenir une architecture parcimonieuse au sens où elle assure une efficacité en pénalisant les structures trop complexes. La structure retenue dans le cadre de cette étude est représentée ci-dessous. Cette dernière est composée de 9 nœuds qui constituent ici les variables explicatives. Ces variables sont composées des valeurs retards à l'ordre de la puissance active, de la puissance apparente, de la tension, du mois et de l'heure de la date d'enregistrement transformées en variables continues.

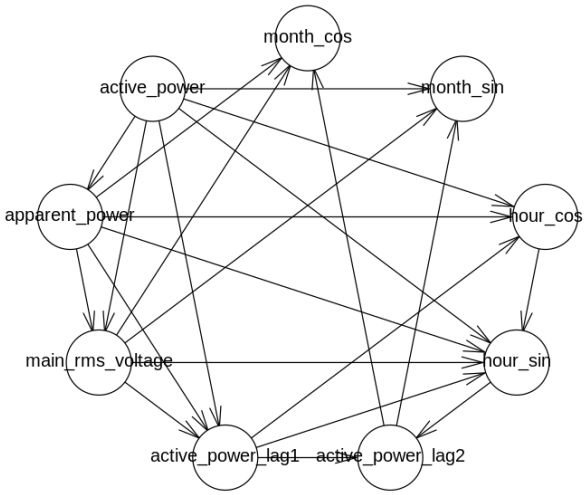

Figure 15 : Architecture du Réseau Bayésien Dynamique

4.2.3 Estimation des paramètres du modèles

Nœud	Parents	Coefficients	Écart-type
Active_power	–	-0.0518	0.9624
Apparent_power	Intercept Active_power	0.0015 0.9947	0.1223
Main_rms_voltage	Intercept Active_power Apparent_power	0.0093 -0.7883 0.5159	0.9514
Active_power_lag1	Intercept Active_power Apparent_power Main_rms_voltage	312.2728 -255.1865 376.0574 -27.0074	192.0572

Active_power_lag2	Intercept Active_power_lag1 Hour_sin	161.7154 0.4717 -58.6096	194.7433
Hour_sin	Intercept Active_power Apparent_power Main_rms_voltage Active_power_lag1 Hour_cos	0.1755 -0.3697 0.3280 0.2217 -0.0006 -0.0570	0.6374
Hour_cos	Intercept Active_power Apparent_power Active_power_lag1	0.0263 0.2043 -0.3534 -0.0001	0.6884
Month_sin	Intercept Active_power Main_rms_voltage Active_power_lag2	-0.0322 0.0527 0.0549 0.0002	0.6872
Month_cos	Intercept Apparent_power Main_rms_voltage Active_power_lag2	-0.2876 0.0682 0.0454 0.0003	0.6827

Tableau 9 : Estimation des paramètres du réseaux bayésien dynamique : Foyer 1

Nœud	Parents	Coefficients	Écart-type
Active_power	–	0.0206	1.0060
Apparent_power	Intercept Active_power Hour_cos Month_sin	-0.0027 0.9945 -0.0188 0.0262	0.0774
Main_rms_voltage	Intercept Active_power Hour_sin Month_sin	-0.1181 -0.3665 -0.2965 0.4394	0.8934
Active_power_lag1	Intercept Apparent_power Hour_cos	301.7256 121.6126 19.6213	232.2229
Active_power_lag2	Intercept Main_rms_voltage Active_power_lag1 Hour_sin Hour_cos	196.3457 21.0507 0.3615 -92.3134 36.6699	222.0166

Hour_sin	Intercept Active_power Apparent_power Active_power_lag1	0.2515 -0.7082 0.5232 -0.0008	0.6187
Hour_cos	Intercept Active_power	0.0050 -0.0811	0.7028
Month_sin	Intercept	0.0529	0.5276
Month_cos	Intercept Apparent_power Main_rms_voltage Active_power_lag2 Hour_sin	-0.8365 0.0327 0.0385 0.0001 0.0280	0.1968

Tableau 10 : Estimation des paramètres du réseaux bayésien dynamique : Foyer 2

Nœud	Parents	Coefficients	Écart-type
Active_power	Intercept Main_rms_voltage Active_power_lag1 Hour_cos	-0.8509 -0.0939 0.0015 -0.0783	0.7866
Apparent_power	Intercept Active_power Active_power_lag1 Hour_sin Hour_cos Month_sin Month_cos	-0.0179 0.9981 -0.00001 -0.0088 -0.0174 -0.0433 -0.0115	0.0488
Main_rms_voltage	Intercept Hour_cos Month_sin Month_cos	-0.4520 0.5276 -0.5743 0.4823	0.8327
Active_power_lag1	Intercept Month_cos	545.7480 -67.1559	384.0384
Active_power_lag2	Intercept Active_power_lag1 Hour_sin Hour_cos	285.8123 0.4986 -98.8183 63.4435	305.8126
Hour_sin	Intercept Active_power Main_rms_voltage Active_power_lag1 Hour_cos	0.2394 -0.1954 -0.1275 -0.0005 0.0532	0.6214
Hour_cos	Intercept	-0.0003	0.7077
Month_sin	Intercept	-0.7867	0.2153

Month_cos	Intercept Month_sin	-1.6196 -1.6124	0.3019

Tableau 11 : Estimation des paramètres du réseaux bayésien dynamique : Foyer 5

Les paramètres de ces réseaux bayésiens s'interprètent de façon très particulière. Ils permettent de mettre en évidence l'influence probabiliste entre les variables du système électrique, telles que la puissance active, la puissance apparente, la tension, ainsi que des composantes temporelles comme l'heure ou le mois.

Pour illustrer ces propos, on a utilisé le nœud « apparent_power » du foyer 5 dont les arrêts sont reliés à 6 autres variables :

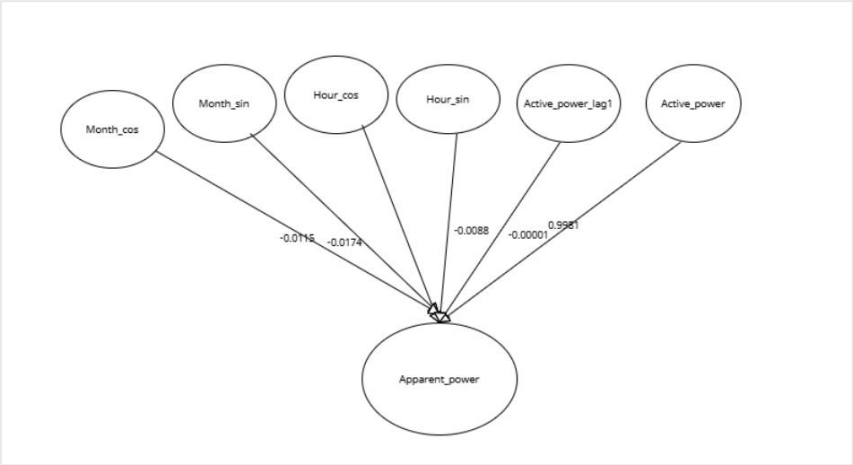

Figure 16 : Schéma du nœud « apparent power » du foyer 5

Comme on peut le constater la coefficient associé à la puissance active vaut 0.9981 ce qui est très proche de 1. Par conséquent, on peut conclure la puissance apparent peut-être entièrement modélisée par la puissance active. Cette observation théorique est même physiquement avérée contenu de la relation en ces deux grandeurs.

4.2.4 Résultats et Discussion

Métrique	Foyer 1	Foyer 2	Foyer 5
Log-Likehood	-245947	-48271.58	-44230.11
AIC	491974	96619.16	88538.22
BIC	492271.6	492271.6	88765.92
MAE	0.8011749	0.8011749	0.5263825
MAPE	211.6339	211.6339	128.8375
RMSE	1.144049	1.144049	0.891147

Tableau 12 : Performances des réseaux bayésiens dynamiques

L'évaluation prédictive des réseaux bayésiens dynamique indiquent une meilleure performance du modèle appliqué au foyer 5, comparativement aux foyers 1 et 2. En effet, les indicateurs d'ajustement statistique tels que la log-vraisemblance, l'AIC et le BIC présentent des valeurs plus faibles pour le foyer 5, traduisant une meilleure adéquation du modèle aux données observées.

Sur le plan prédictif, les métriques d'erreur telles que le MAE, le MAPE et le RMSE confirment cette tendance, avec des erreurs significativement plus faibles pour le foyer 5, indiquant une précision accrue des prédictions. Ces résultats suggèrent que la structure de consommation du foyer 5 est plus régulière ou mieux captée par le modèle bayésien, ce qui facilite son apprentissage et améliore ses performances.

5 CONCLUSION

En somme, la prévision de la consommation d'énergie est l'une des méthodes les plus privilégiées pour atteindre l'efficience énergétique. Cette étude a permis de montrer qu'en raison de son caractère non linéaire et de sa nature multifactorielle, plusieurs modèles mathématiques ont été développés. Parmi ces modèles les plus populaires sont les méthodes d'analyse de données qui regroupent les modèles de régression linéaire, les modèles de régression non linéaires, les séries temporelles, les méthodes d'apprentissage automatiques, les méthodes d'apprentissage profond et enfin les méthodes hybrides.

Ce travail s'est appuyé sur cette dernière catégorie de modèle en utilisant notamment les réseaux bayésiens dynamiques. Les résultats ont montré qu'en termes de log-vraisemblance, le modèle ARIMA doit être préféré à celui du réseau probabiliste. Cependant, en ce qui concerne les prévisions, les erreurs sont significativement plus faibles avec le modèle bayésien, notamment pour le foyer 5 où la précision des prévisions est notablement améliorée, comme en témoigne le MAPE réduit à 128,8375 contre 211,6339 pour ARIMA.

Ainsi, le modèle bayésien semble donc plus robuste et précis pour les prévisions de consommation d'énergie, en particulier pour les foyers avec des comportements de consommation plus réguliers et moins influencés par des facteurs externes complexes. En revanche, le modèle ARIMA, bien qu'efficace pour capturer les tendances à court terme, semble moins performant lorsqu'il s'agit de prédire précisément la consommation sur une plus longue période ou lorsque des effets non linéaires et des dépendances temporelles complexes entrent en jeu.

Pour améliorer les prévisions de la consommation d'énergie, plusieurs pistes peuvent être envisagées. On peut citer l'intégration de nouvelles variables exogènes, le changement du critère de score utilisé dans l'apprentissage du réseau et enfin le passage vers un réseau non gaussien.

6 BIBLIOGRAPHIQUE

- **Papon, P. (2024).** *World Energy Outlook 2023*. Paris : Agence internationale de l'énergie (AIE). *Futuribles*, (2), 114–117.
- **Ritchie, H., Roser, M., & Rosado, P.** (2024). *Energy production and consumption*. Our World in Data.
- **Amasyali, K., & El-Gohary, N. M**. (2018). A review of data-driven building energy consumption prediction studies. *Renewable and Sustainable Energy Reviews, 81*, 1192–1205.
- **Al-Garni, A. Z., Zubair, S. M., & Nizami, J. S**. (1994). A regression model for electric energy-consumption forecasting in Eastern Saudi Arabia. *Energy, 19*(10), 1043–1049.
- **Tunç, M., Çamdali, Ü., & Parmaksizoğlu, C.** (2006). Comparison of Turkey's electrical energy consumption and production with some European countries and optimization of future electrical power supply investments in Turkey. *Energy Policy, 34*(1), 50–59.
- **Hao, Y., Zhang, Z. Y., Liao, H., & Wei, Y. M.** (2015). China's farewell to coal: A forecast of coal consumption through 2020. *Energy Policy, 86*, 444–455.
- **Chujai, P., Kerdprasop, N., & Kerdprasop, K.** (2013, March). Time series analysis of household electric consumption with ARIMA and ARMA models. In *Proceedings of the International Multiconference of Engineers and Computer Scientists* (Vol. 1, pp. 295–300).
- **Yasmeen, F., & Sharif, M.** (2014). Forecasting electricity consumption for Pakistan. *International Journal of Emerging Technology and Advanced Engineering, 4*(4), 496–503.
- **Wang, Z., Srinivasan, R., & Wang, Y.** (2016). Homogeneous ensemble model for building energy prediction: A case study using ensemble regression tree. In *Proceedings of the 2016 ACEEE Summer Study on Energy Efficiency in Buildings*, Pacific Grove, CA, USA, 21–26.
- **Dong, B., Cao, C., & Lee, S. E.** (2005). Applying support vector machines to predict building energy consumption in tropical region. *Energy and Buildings, 37*(5), 545–553.
- **Ahmad, A. S., Hassan, M. Y., Abdullah, M. P., Rahman, H. A., Hussin, F., Abdullah, H., & Saidur, R.** (2014). A review on applications of ANN and SVM for building electrical energy consumption forecasting. *Renewable and Sustainable Energy Reviews, 33*, 102–109.
- **Azadeh, A., Ghaderi, S. F., & Sohrabkhani, S.** (2008). Annual electricity consumption forecasting by neural network in high energy consuming industrial sectors. *Energy Conversion and Management, 49*(8), 2272–2278.
- **Amalou, I., Mouhni, N., & Abdali, A**. (2022). Multivariate time series prediction by RNN architectures for energy consumption forecasting. *Energy Reports, 8*, 1084–1091.
- **Kaytez, F., Taplamacioglu, M. C., Cam, E., & Hardalac, F.** (2015). Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines. *International Journal of Electrical Power & Energy Systems, 67*, 431–438.

- **Aulia, H., Syaharuddin, S., Mandailina, V., Gervas, H. E., & Ashraf, H.** (2024). Probabilistic forecasting of energy consumption using Bayesian dynamic linear models. *Aceh International Journal of Science and Technology, 13*(1), 68–78.
- **Zhang, Q., Yan, H., & Liu, Y.** (2024). Power generation forecasting for solar plants based on dynamic Bayesian networks by fusing multi-source information. *Renewable and Sustainable Energy Reviews, 202*, 114691.
- **Singh, S., & Yassine, A**. (2018). Big data mining of energy time series for behavioral analytics and energy consumption forecasting. *Energies, 11*(2), 452.
- **Kelly, J., & Knottenbelt, W.** (2015). The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes. *Scientific data*, *2*(1), 1-14.

7 ANNEXES

7.1 EXPLORATION DES DONNEES : GRAPHIQUES

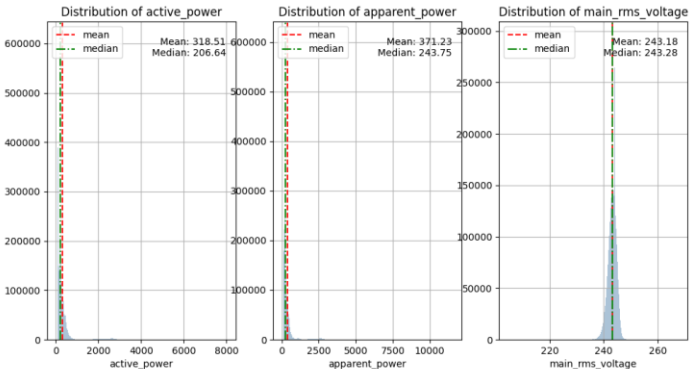

Figure 17 : Histogramme de la consommation d'électricité - Foyer 1

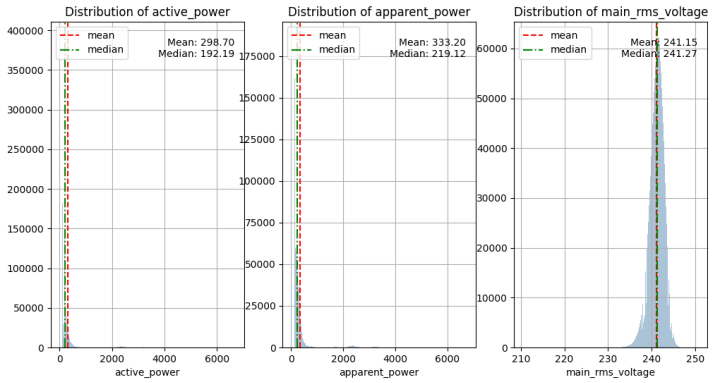

Figure 18 : Histogramme de la consommation d'électricité - Foyer 2

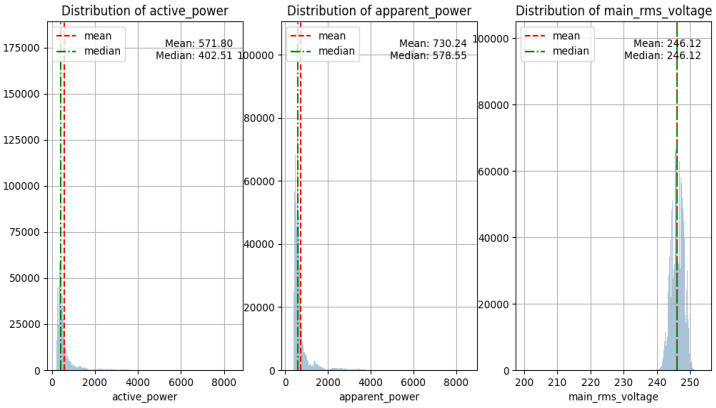

Figure 19 : Histogramme de la consommation d'électricité - Foyer 5

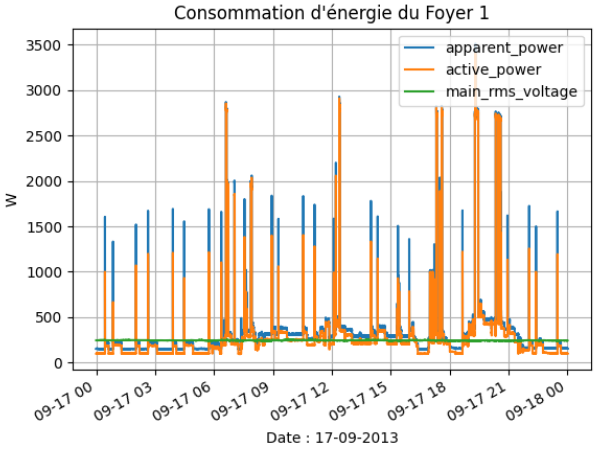

Figure 20 : Consommation d'énergie du foyer 1 : filtré

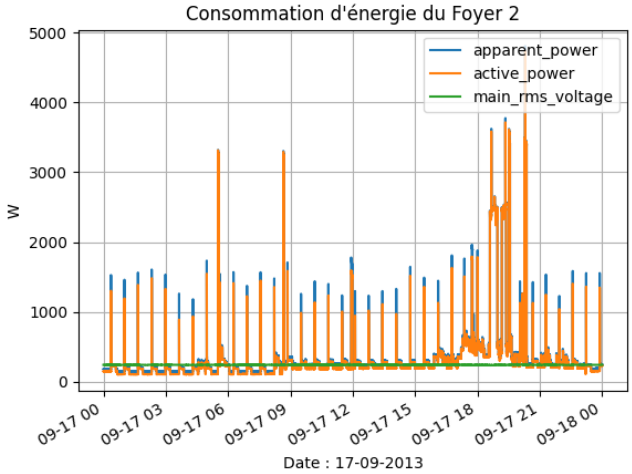

Figure 21 : Consommation d'énergie du foyer 2 : filtré

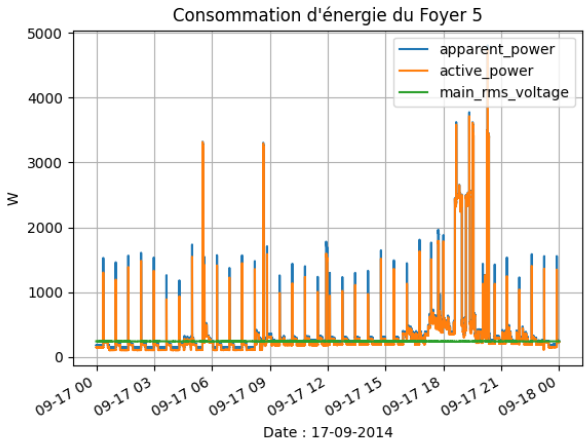

Figure 22 : Consommation d'énergie du foyer 5 - filtré

Figure 23 : Décomposition de la série - Foyer 1

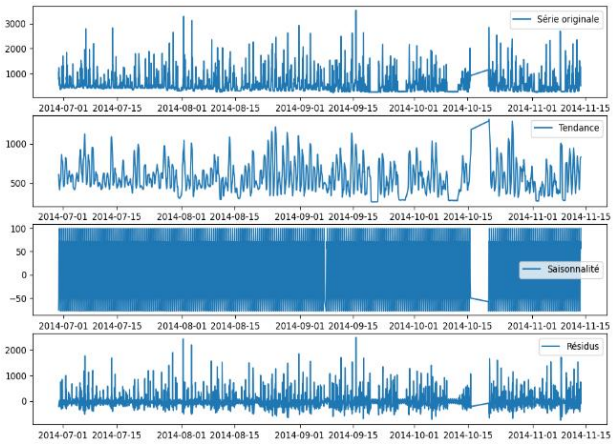

Figure 24 : Décomposition de la série - Foyer 2

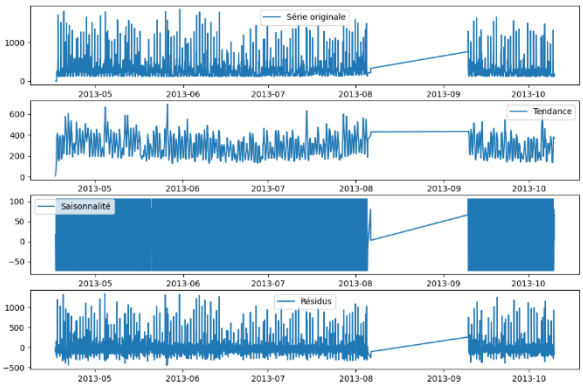

Figure 25 : Décomposition de la série - Foyer 5

7.2 PROGRAMME : EXPLORATION DES DONNEE

```
## **I - Chemin vers le Google Drive**

# Code pour accéder aux dossiers Google Drive
from google.colab import drive
drive.mount('/content/drive')
## **II - Importation des librairies**
# Importer les librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
## **III - Paramétrages des librairies**
# PARAMETRES DE PANDAS

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# PARAMETRES DE MATPLOTLIB

## Line

plt.rcParams["lines.linewidth"] = 1.5
plt.rcParams["lines.linestyle"] = "-" # https://matplotlib.org/stable/gal-
lery/lines_bars_and_markers/linestyles.html
plt.rcParams["lines.color"] = "C0"
plt.rcParams["lines.marker"] = "none"
plt.rcParams["lines.markeredgewidth"] = 1.0
plt.rcParams["lines.markersize"] = 6
plt.rcParams["lines.markerfacecolor"] = "auto"
plt.rcParams["lines.markeredgecolor"] = "auto"

## Boxplot

plt.rcParams["boxplot.notch"] = False
plt.rcParams["boxplot.vertical"] = True
plt.rcParams["boxplot.whiskers"] = 1.5
plt.rcParams["boxplot.bootstrap"] = None
plt.rcParams["boxplot.patchartist"] = False
plt.rcParams["boxplot.showmeans"] = True
plt.rcParams["boxplot.showcaps"] = True
plt.rcParams["boxplot.showbox"] = True
plt.rcParams["boxplot.showfliers"] = True
plt.rcParams["boxplot.meanline"] = True
```

```
plt.rcParams["boxplot.flierprops.color"] = "black"
plt.rcParams["boxplot.flierprops.marker"] ="o"
plt.rcParams["boxplot.flierprops.markerfacecolor"] = "none"
plt.rcParams["boxplot.flierprops.markeredgecolor"] = "black"
plt.rcParams["boxplot.flierprops.markeredgewidth"] = 1.0
plt.rcParams["boxplot.flierprops.markersize"] = 6
plt.rcParams["boxplot.flierprops.linestyle"] = "none"
plt.rcParams["boxplot.flierprops.linewidth"] = 1.0

plt.rcParams["boxplot.boxprops.color"] = "black"
plt.rcParams["boxplot.boxprops.linewidth"] = 1.0
plt.rcParams["boxplot.boxprops.linestyle"] = "-"

plt.rcParams["boxplot.whiskerprops.color"] = "black"
plt.rcParams["boxplot.whiskerprops.linewidth"] = 1.0
plt.rcParams["boxplot.whiskerprops.linestyle"] = "-"

plt.rcParams["boxplot.capprops.color"] = "black"
plt.rcParams["boxplot.capprops.linewidth"] = 1.0
plt.rcParams["boxplot.capprops.linestyle"] = "-"

plt.rcParams["boxplot.medianprops.color"] = "C1"
plt.rcParams["boxplot.medianprops.linewidth"] = 1.0
plt.rcParams["boxplot.medianprops.linestyle"] = "-"

plt.rcParams["boxplot.meanprops.color"] = "C2"
plt.rcParams["boxplot.meanprops.marker"] = "^"
plt.rcParams["boxplot.meanprops.markerfacecolor"]= "C2"
plt.rcParams["boxplot.meanprops.markeredgecolor"]= "C2"
plt.rcParams["boxplot.meanprops.markersize"]= 6
plt.rcParams["boxplot.meanprops.linestyle"]= "--"
plt.rcParams["boxplot.meanprops.linewidth"]= 1.0


## Axes

plt.rcParams["axes.facecolor"] = "white"
plt.rcParams["axes.edgecolor"] = "black"
plt.rcParams["axes.linewidth"] = 0.8
plt.rcParams["axes.grid"] = True
plt.rcParams["axes.grid.axis"] = "both"
plt.rcParams["axes.grid.which"] = "major"
plt.rcParams["axes.titlelocation"] = "center"
plt.rcParams["axes.titlesize"] ="large"
plt.rcParams["axes.titleweight"] = "normal"
plt.rcParams["axes.titlecolor"] = "auto"

plt.rcParams["axes.titley"] = None
```

```
plt.rcParams["axes.titlepad"] = 6.0
plt.rcParams["axes.labelsize"] = "medium"
plt.rcParams["axes.labelpad"] = 4.0
plt.rcParams["axes.labelweight"] = "normal"
plt.rcParams["axes.labelcolor"] = "black"
plt.rcParams["axes.axisbelow"] = "line"

## Grid

plt.rcParams["grid.color"] = "#b0b0b0"
plt.rcParams["grid.linestyle"] = "-"
plt.rcParams["grid.linewidth"] = 0.8
plt.rcParams["grid.alpha"] = 1.0

## Figure

plt.rcParams["figure.figsize"] = 6.4, 4.8
plt.rcParams["figure.dpi"] = 100
plt.rcParams["figure.facecolor"] = "white"  # Paramètre à modifier en gris
plt.rcParams["figure.edgecolor"] = "white"

## Legend

plt.rcParams["legend.loc"] = "upper right" # best
plt.rcParams["legend.frameon"] = True
plt.rcParams["legend.framealpha"] = 0.8
plt.rcParams["legend.facecolor"] = "inherit"
plt.rcParams["legend.fancybox"] = True
plt.rcParams["legend.shadow"] = False
plt.rcParams["legend.numpoints"] = 1
plt.rcParams["legend.scatterpoints"] = 1
plt.rcParams["legend.markerscale"] = 1.0
plt.rcParams["legend.fontsize"] = "medium"

## Image

plt.rcParams["image.cmap"] = "viridis" # # A colormap name (plasma, magma,
etc.) https://matplotlib.org/stable/users/explain/colors/colormaps.html

## Saving figure

plt.rcParams["savefig.format"] = "png"  # {png, ps, pdf, svg}
plt.rcParams["savefig.directory"] = "~"
plt.rcParams["savefig.orientation"] = "portrait"

# PARAMETRES DE SEABORN
color_palette = sns.color_palette("crest", as_cmap=True)
```

```
# PARAMETRES OS

print(os.getcwd())
os.makedirs("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/data_preprocessed", exist_ok=True)
os.makedirs("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics", exist_ok=True )

## **IV - Chargemenet des bases de données**
# arguments pour le chargement des données
sep_df = "\s+" # <<\t>> dans d'autres cas.
col_df = ["unix", "active_power", "apparent_power", "main_rms_voltage"]
# Charger de la base de données à l'aide de pandas
house_1 = pd.read_csv(r"/content/drive/MyDrive/ColabNotebooks/Da-
taset/house_1/mains.dat", sep=sep_df, names=col_df)
house_2 = pd.read_csv(r"/content/drive/MyDrive/ColabNotebooks/Da-
taset/house_2/mains.dat", sep=sep_df, names=col_df)
house_5 = pd.read_csv(r"/content/drive/MyDrive/ColabNotebooks/Da-
taset/house_5/mains.dat", sep=sep_df, names=col_df)
# head.dataframe de house_1_brute : Foye 1
house_1.head(5)
# head.dataframe de house_2_brute : Foyer 2
house_2.head(5)
# head.dataframe de house_3_brute : Foyer 5
house_5.head(5)
## **V - Manipulation des bases de données**
### A - Fonction pour créer les variables de type datetime(s)
# fonction pour créer des variables dates, Année, Mois, Jours et Séconde
def fct_unix_to_date(dataframe, variable="unix", house_i=1):

    """
    description
    ------------
    fonction qui permet de créer les variables "unix_year", "unix_month",
"unix_day","unix_minute", "unix_second", "unix_microsecond"
    à partir de la variable unix et d'organiser les colonnes.

    arguments
    ----------
    dataframe : La base de données
    variable : La variable correspondante à l'unix stamp (1970-01-01
00:00:00 UTC)
    origin_date : house_1 : 2012-11-09 house_2 2013-02-17 house_3 2013-02-
27 house_4 : 2013-03-09 house_5: 2014-06-29 ( format %year-%Month-
%day  %hour:%minute:%second.microsecond")

    Optional Parameters
    --------------------
```

```
    #  Timestamps en fonction du premier enregistrement
     if house_i == 1:
        origin_date = "2012-11-09  00:00:00"
    elif house_i ==2 :
        origin_date = "2013-02-17  00:00:00"
    elif house_i ==3:
        origin_date = "2013-02-27  00:00:00"
    elif house_i == 4:
        origin_date = "2013-03-09  00:00:00 "
    else :
        origin_date = "2014-06-29  00:00:00"

    # Convertir origin_date en datetime
    origin_datetime = pd.to_datetime(origin_date)

    """


    # Trier le DataFrame par la colonne UNIX
    dataframe = pd.DataFrame(dataframe)
    dataframe = dataframe.sort_values(by=variable, ascending=True)
    New_variable = ["unix_year", "unix_month", "unix_day",
"unix_hour","unix_minute", "unix_second", "unix_microsecond"]

    # Créer une variable unix au format date à partir de la variable unix
    dataframe.index = pd.to_datetime(dataframe[variable], unit="s" ) #
origin=pd.Timestamp(origin_date) | format='%Y-%m-%d %H:%M:%S.%f

    # Créer des variables year, month, day et second
    dataframe[New_variable[0]] = dataframe.index.year
    dataframe[New_variable[1]] = dataframe.index.month
    dataframe[New_variable[2]] = dataframe.index.day
    dataframe[New_variable[3]] = dataframe.index.hour
    dataframe[New_variable[4]] = dataframe.index.minute
    dataframe[New_variable[5]] = dataframe.index.second
    dataframe[New_variable[6]] = dataframe.index.microsecond


    # Réorganiser la base de données
    all_new_variable = ["unix", "unix_year", "unix_month",
"unix_day","unix_hour","unix_minute", "unix_second", "unix_microsecond",
"active_power", "apparent_power", "main_rms_voltage"]
    dataframe_ordered = dataframe[all_new_variable]
    return dataframe_ordered
# Création des bases de données avec les variables unix_date(s)
house_1_new = fct_unix_to_date(house_1, variable="unix", house_i=1)
house_2_new = fct_unix_to_date(house_2, variable="unix", house_i=2)
house_5_new = fct_unix_to_date(house_5, variable="unix", house_i=5)
```

```
# head.dataframe de house_1_new : Foyer 1
print(house_1_new.info())
print("--------------")
house_1_new.head(5)
# head.dataframe de house_2_new : Foyer 2
print(house_2_new.info())
print("--------------")
house_2_new.head(5)
# head.dataframe de house_5_new : Foyer 5
print(house_5_new.info())
print("--------------")
house_5_new.head(5)
# Sauvegarde des fichiers house_i_new
house_1_new.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/data_preprocessed/house_1_new.csv", index=False)
house_2_new.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/data_preprocessed/house_2_new.csv", index=False)
house_5_new.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/data_preprocessed/house_5_new.csv", index=False)
### B - Traitement des valeurs nulles et des valeurs manquantes
# Vérifier si house_1_new contient des valeurs manquantes
print(house_1_new.isna().sum())
print("----------x------------")
# Vérifier si house_1_new contient des valeurs nulles
print(house_1_new.isnull().sum())
# Vérifier si house_2_new contient des valeurs manquantes
print(house_2_new.isna().sum())
print("----------x------------")
# Vérifier si house_1_new contient des valeurs nulles
print(house_2_new.isnull().sum())
# Vérifier si house_5_new contient des valeurs manquantes
print(house_5_new.isna().sum())
print("----------x------------")
# Vérifier si house_1_new contient des valeurs nulles
print(house_5_new.isnull().sum())
control_drop = False
if control_drop == True:
  # Suppression des valeurs manquantes
  house_1_new.dropna(inplace=True, axis=0)
  house_2_new.dropna(inplace=True, axis=0)
  house_5_new.dropna(inplace=True, axis=0)
### C - Fonction pour convetir les variables en float et int
# fonction pour convertir les variables en float

def fct_convert_to_numeric(dataframe):
    """
    description
    ----------
```

```
    Cette fonction permet de convertir les variables unix_type(s) en format
numérique. Cela facilite la manipulation des données
    argument
    --------
    dataframe : La base de données avec les dates à jour
    """
    df = pd.DataFrame(dataframe)
    # Convertir les variables de la liste List_col en variable numérique
    List_col = ['unix_year', 'unix_month', 'unix_day', "unix_hour",
'unix_minute', 'unix_second', 'unix_microsecond', 'active_power', 'appar-
ent_power', 'main_rms_voltage']

    for j in List_col:
        df[j] = pd.to_numeric(df[j], errors="coerce", downcast="integer")

        if j in List_col[:7]:
            df[j] = df[j].astype("int32")
            # Afficer le type de chaque variables
            print(f" * La variable : {j} est doublement convertie en :
__{df[j].dtype}")
        else:
            print(f" - La variable : {j} est convertie uniquement en :
__{df[j].dtype}")
    return df

# Création des bases de données avec les variables numérique float et int

house_1_new_numeric = fct_convert_to_numeric(house_1_new)
print("-------------x-----------------")
house_2_new_numeric = fct_convert_to_numeric(house_2_new)
print("-------------x-----------------")
house_5_new_numeric = fct_convert_to_numeric(house_5_new)
# head.dataframe de house_1_new_numeric : Foyer 1
house_1_new_numeric.head(5)
# head.dataframe de house_2_new_numeric : Foyer 2
house_2_new_numeric.head(5)
# head.dataframe de house_5_new_numeric : Foyer 5
house_5_new_numeric.head(5)
# Sauvegarde des fichiers house_i_new_numeric
house_1_new_numeric.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataM-
ining__/data_preprocessed/house_1_new_numeric.csv", index=True)
house_2_new_numeric.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataM-
ining__/data_preprocessed/house_2_new_numeric.csv", index=True)
house_5_new_numeric.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataM-
ining__/data_preprocessed/house_5_new_numeric.csv", index=True)
# Fonction pour créer la variable unix_id en fonction de la variable pério-
dicité
def fct_creation_unix_id_var(dataframe, periodicite="unix_day"):
```

```
    '''
    description:
    ------------
    Cette fonction sert à créer un dataframe en fonction de la fréquence
choisie.

    argument:
    ---------
    dataframe : Le dataframe sélectionné.
    periodicite : La périodicité choisie pour la création du dataframe.

    option
    ------
    La fonction str_format = '_'.join(['{}'] * size_new_list) remplace le
code suivant :
    '''
    df = pd.DataFrame(dataframe)
    List_col = df.columns.tolist()
    Index_perdiocite = List_col.index(periodicite)
    New_List_col = List_col[1:Index_perdiocite + 1]
    size_new_list = len(New_List_col)

    str_format = '_'.join(['{}'] * size_new_list)

    df["unix_id"] = df[New_List_col].apply(lambda row: str_format.for-
mat(*row), axis=1)
    return df



# Création des dataframes avec la variable unix_id en fonction de la pério-
dicité hour
house_1_new_numeric_preprocessed = fct_creation_unix_id_var(house_1_new_nu-
meric, periodicite="unix_hour")
house_2_new_numeric_preprocessed = fct_creation_unix_id_var(house_2_new_nu-
meric, periodicite="unix_hour")
house_5_new_numeric_preprocessed = fct_creation_unix_id_var(house_5_new_nu-
meric, periodicite="unix_hour")
# head.dataframe de house_1_new_numeric_preprocessed : Foyer 1
house_1_new_numeric_preprocessed.head(5)
# head.dataframe de house_2_new_numeric_preprocessed : Foyer 2
house_2_new_numeric_preprocessed.head(5)
# head.dataframe de house_5_new_numeric_preprocessed : Foyer 5
house_5_new_numeric_preprocessed.head(5)
# Sauvegarde des fichiers house_i_n_new_numeric_preprocesse
```

```
house_1_new_numeric_preprocessed.to_csv("/content/drive/MyDrive/Colab-
Notebooks/1__DataMining__/data_preprocessed/house_1_new_numeric_prepro-
cessed.csv", index=True)
house_2_new_numeric_preprocessed.to_csv("/content/drive/MyDrive/Colab-
Notebooks/1__DataMining__/data_preprocessed/house_2_new_numeric_prepro-
cessed.csv", index=True)
house_5_new_numeric_preprocessed.to_csv("/content/drive/MyDrive/Colab-
Notebooks/1__DataMining__/data_preprocessed/house_5_new_numeric_prepro-
cessed.csv", index=True)

## **VI - Statistiques descriptives**
### A - Fonction pour dresser l'inventaire des bases de données
# L'inventaire doit se faire en utilisant la variable unix_date et les
opération pd.datetime
def fct_inventaire_database(dataframe, name):
    '''
    description
    -----------
    fonction pour dresser l'inventaire de la base de données. Elle per-
met d'obtenir un aperçu du nombre de variable, le nombre d'année étu-
diées...
    arguments
    ---------
    dataframe : La base de données
    name : Le nom de la base de données en format str
    '''

    df = pd.DataFrame(dataframe)
    List_cols = ["unix_year", "unix_month", "unix_day",
"unix_hour","unix_minute","unix_second", "unix_microsecond"]

    start_date = df.index.min()
    end_date = df.index.max()
    delta = end_date - start_date

    days = delta.days
    seconds = delta.seconds

    years = days // 365
    months = (days % 365) // 30
    hours = seconds // 3600
    minutes = (seconds % 3600) // 60


    #   Création d'un dictionnaire contenant les informations clés
    Matrice = {
        'dataframe_name_': name,
        'rows_no_': df.shape[0],
```

```
        'cols_no_': df.shape[1],
        'start_date_': start_date,
        'end_date_': end_date,
        'years_no_': years,
        "mois_no_": months,
        'days_no_': days,
        'hours_no_': days*24

    }

    return Matrice

# Liste contenant les dataframes et leurs noms
listes_dataframes = [
    (house_1_new_numeric_preprocessed, 'house_1_new_numeric_prepro-
cessed'),
    (house_2_new_numeric_preprocessed, 'house_2_new_numeric_prepro-
cessed'),
    (house_5_new_numeric_preprocessed, 'house_5_new_numeric_prepro-
cessed'),
]

# Création du dataframe qui contient l'inventaire des bases de données
resultats = []
for data, name in listes_dataframes:
    resultats.append(fct_inventaire_database(data, name))
resultats_df = pd.DataFrame(resultats)
resultats_df

### B - Fonction pour résumer les statistiques descriptives
def fct_matrice_descriptive(dataframe, columns=["active_power", "appa-
rent_power", "main_rms_voltage"]):
    """
    description
    -----------
    fonction pour calculer les statistiques descriptives en fonction du
niveau de consommation

    arguments
    -----------
    dataframe : Base de données de type pandas

    """
    df = pd.DataFrame(dataframe)
    df_filtred = df[columns]

    resultats = {}
```

```
    for i in df_filtred.select_dtypes(include=['float', 'inte-
ger']).columns:
        matrice = {
            'Max': round(df_filtred[i].max(),2),
            'Min': round(df_filtred[i].min(),2),
            'Moyenne': round(df_filtred[i].mean(),2),
            'Mediane': round(df_filtred[i].median(),2),
            'Q1': round(df_filtred[i].quantile(0.25),2),
            'Q3': round(df_filtred[i].quantile(0.75),2),
            'Ecart-type': round(df_filtred[i].std(),2),
            'Variance': round(df_filtred[i].var(),2),
            'Coefficient de variation': round(((df_fil-
tred[i].std()/df_filtred[i].mean()) * 100),2),
            'coef_skewness': round(df_filtred[i].skew(),2),
            'coef_kurtosis': round(df_filtred[i].kurtosis(),2),
            'Valeurs nulles': df_filtred[i].isnull().sum()
        }
        resultats[i] = matrice
    resultats = pd.DataFrame(resultats)
    return resultats

# Création des dataframes pour afficher les statistiques descriptives
stats_desc_house_1 = fct_matrice_descriptive(house_1_new_numeric_pre-
processed)
stats_desc_house_2 = fct_matrice_descriptive(house_2_new_numeric_pre-
processed)
stats_desc_house_5 = fct_matrice_descriptive(house_5_new_numeric_pre-
processed)

# Statistiques descriptives de house_1 : Foyer 1
stats_desc_house_1

# Statistiques descriptives de house_2 : Foyer 2
stats_desc_house_2

# Sauvegarder les tables de statistiques descriptives
stats_desc_house_1.to_csv("/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_1_stats_des.csv", in-
dex=True)
stats_desc_house_2.to_csv("/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_2_stats_des.csv", in-
dex=True)
stats_desc_house_5.to_csv("/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_5_stats_des.csv", in-
dex=True)

## **VII - Visualisation des données : Période Globale**
```

```
### A - Plot

house_1_new_numeric_preprocessed[["active_power"]].plot(title="Consom-
mation d'énergie : Foyer 1")
plt.xlabel("Date")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_energy_consumption_global.png")

house_2_new_numeric_preprocessed[["active_power"]].plot(title="Consom-
mation d'énergie : Foyer 2")
plt.xlabel("Date")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_energy_consumption_global.png")

house_5_new_numeric_preprocessed[["active_power"]].plot(title="Consom-
mation d'énergie : Foyer 5")
plt.xlabel("Date")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_energy_consumption_global.png")

#### A.2 - Fonction pour regrouper les données en fonction de la pério-
dicité par  : Heure

# Fonction pour filtrer les données à la date indiquée. ( Date de réfé-
rence ~ 20-04-2014 )

def fct_data_groupby_timeserie(dataframe, freq_target="H"):

    '''
    description:
    ------------
    Cette fonction permet de grouper les données selon un critère de
périodicité défini.

    argument:
    ---------
    dataframe : La base de données contenant les variables de type da-
tetime(s) et variables explicatives.
```

```
    period_target : Périodicité utilisée pour filtrer les données
freq=["YE", "YS", "ME", "MS", "W", "D", "H", "min", "s"].

    '''
    # Création de la variable (floor) utilisée pour grouper les données
    df = pd.DataFrame(dataframe)
    df["unix_id_group"] = df.index.floor(freq_target)

     # Création du dataframe groupé en fonction de la périodicité de
filtre
    df_groupby = df.groupby("unix_id_group").agg({
        "active_power": "mean",
        "apparent_power": "mean",
        "main_rms_voltage": "mean"

    }).reindex()

    df_groupby.reindex(df["unix_id_group"].tolist())
    return df_groupby

# Création des dataframe en utilisant la fonction pour regrouper les
données en heure.
house_1_new_numeric_preprocessed_H = fct_data_groupby_timeserie(data-
frame=house_1_new_numeric_preprocessed, freq_target="H")
house_2_new_numeric_preprocessed_H = fct_data_groupby_timeserie(data-
frame=house_2_new_numeric_preprocessed, freq_target="H")
house_5_new_numeric_preprocessed_H = fct_data_groupby_timeserie(data-
frame=house_5_new_numeric_preprocessed, freq_target="H")

# head.dataframe par heure : Foyer 2
house_1_new_numeric_preprocessed_H.head(5)

# head.dataframe par heure : Foyer 2
house_2_new_numeric_preprocessed_H.head(5)

# head.dataframe par heure : Foyer 5
house_5_new_numeric_preprocessed_H.head(5)

# Création des dataframes pour afficher les statistiques descriptives
stats_desc_house_1_H = fct_matrice_descriptive(house_1_new_numeric_pre-
processed_H)
stats_desc_house_2_H = fct_matrice_descriptive(house_2_new_numeric_pre-
processed_H)
stats_desc_house_5_H = fct_matrice_descriptive(house_5_new_numeric_pre-
processed_H)

# Statistiques descriptives en heure : Foyer 1
stats_desc_house_1_H
```

```
# Statistiques descriptives en heure : Foyer 2
stats_desc_house_2_H

# Statistiques descriptives en heure : Foyer 5
stats_desc_house_5_H

house_1_new_numeric_preprocessed_H[["apparent_power", "active_power",
"main_rms_voltage"]].plot(title="Consommation d'énergie : Foyer 1")
plt.xlabel("Date")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_energy_consumption_global_H.png")

house_2_new_numeric_preprocessed_H[["apparent_power", "active_power",
"main_rms_voltage"]].plot(title="Consommation d'énergie : Foyer 2")
plt.xlabel("Date")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_energy_consumption_global_H.png")

house_5_new_numeric_preprocessed[["apparent_power", "active_power",
"main_rms_voltage"]].plot(title="Consommation d'énergie : Foyer 5")
plt.xlabel("Date")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_energy_consumption_global_H.png")

### B - Boxplot

# Boxplot du Foyer 1 selon une périodicité par heure
plt.figure(figsize=(12,6))
sns.boxplot(data=house_1_new_numeric_preprocessed,
             x="unix_hour",
             y="active_power",
            palette="viridis")

plt.title(f"Consommation d'énergie : Foyer 1")
plt.xlabel("Date")
plt.ylabel("W")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_boxplot_global.png")
plt.grid(True)
```

```
plt.show()

# Boxplot du Foyer 2 selon une périodicité par heure
plt.figure(figsize=(12,6))
sns.boxplot(data=house_2_new_numeric_preprocessed,
            x="unix_hour",
            y="active_power",
            palette="viridis")

plt.title(f"Consommation d'énergie : Foyer 2")
plt.xlabel("Date")
plt.ylabel("W")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_boxplot_global.png")
plt.grid(True)
plt.show()

# Boxplot du Foyer 5 selon une périodicité par heure
plt.figure(figsize=(12,6))
sns.boxplot(data=house_5_new_numeric_preprocessed,
            x="unix_hour",
            y="active_power",
            palette="viridis")

plt.title(f"Consommation d'énergie : Foyer 5")
plt.xlabel("Date")
plt.ylabel("W")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_boxplot_global.png")
plt.grid(True)
plt.show()

# Histogramme de la consommation d'énergie du foyer 1
house_1_groupby_viz_filtred = house_1_new_numeric_preprocessed[["ac-
tive_power", "apparent_power", "main_rms_voltage"]]

fig, axs = plt.subplots(1, 3, figsize=(12, 6))
axs = axs.ravel()

for i, colonne in enumerate(house_1_groupby_viz_filtred.se-
lect_dtypes(include=["float", "integer"]).columns):
    sns.distplot(house_1_groupby_viz_filtred[colonne], ax=axs[i],
bins='auto', hist=True, rug=False, kde=False,
                 color='#2E6A9D')
    axs[i].axvline(house_1_groupby_viz_filtred[colonne].mean(),
color='red', linestyle='--', label='mean')
    axs[i].axvline(house_1_groupby_viz_filtred[colonne].median(),
color='green', linestyle='-.', label='median')
```

```
    median = house_1_groupby_viz_filtred[colonne].median()
    mean = house_1_groupby_viz_filtred[colonne].mean()

    axs[i].text(0.95, 0.95, f'Mean: {mean:.2f}\nMedian: {median:.2f}',
transform=axs[i].transAxes, fontsize=10,
                va='top', ha='right')
    axs[i].legend(loc='upper left')
    axs[i].set_title(f"Distribution of {colonne}")
    axs[i].grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_histplot_global.png")
plt.tight_layout()
plt.show()

# Histogramme de la consommation d'énergie du foyer 2
house_2_groupby_viz_filtred = house_2_new_numeric_preprocessed[["ac-
tive_power", "apparent_power", "main_rms_voltage"]]

fig, axs = plt.subplots(1, 3, figsize=(12, 6))
axs = axs.ravel()

for i, colonne in enumerate(house_2_groupby_viz_filtred.se-
lect_dtypes(include=["float", "integer"]).columns):
    sns.distplot(house_2_groupby_viz_filtred[colonne], ax=axs[i],
bins='auto', hist=True, rug=False, kde=False,
                 color='#2E6A9D')
    axs[i].axvline(house_2_groupby_viz_filtred[colonne].mean(),
color='red', linestyle='--', label='mean')
    axs[i].axvline(house_2_groupby_viz_filtred[colonne].median(),
color='green', linestyle='-.', label='median')

    median = house_2_groupby_viz_filtred[colonne].median()
    mean = house_2_groupby_viz_filtred[colonne].mean()

    axs[i].text(0.95, 0.95, f'Mean: {mean:.2f}\nMedian: {median:.2f}',
transform=axs[i].transAxes, fontsize=10,
                va='top', ha='right')
    axs[i].legend(loc='upper left')
    axs[i].set_title(f"Distribution of {colonne}")
    axs[i].grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_histplot_global.png")
plt.tight_layout()
plt.show()

# Histogramme de la consommation d'énergie du foyer 5
```

```
house_5_groupby_viz_filtred = house_5_new_numeric_preprocessed[["ac-
tive_power", "apparent_power", "main_rms_voltage"]]

fig, axs = plt.subplots(1, 3, figsize=(12, 6))
axs = axs.ravel()

for i, colonne in enumerate(house_5_groupby_viz_filtred.se-
lect_dtypes(include=["float", "integer"]).columns):
    sns.distplot(house_5_groupby_viz_filtred[colonne], ax=axs[i],
bins='auto', hist=True, rug=False, kde=False,
                 color='#2E6A9D')
    axs[i].axvline(house_5_groupby_viz_filtred[colonne].mean(),
color='red', linestyle='--', label='mean')
    axs[i].axvline(house_5_groupby_viz_filtred[colonne].median(),
color='green', linestyle='-.', label='median')

    median = house_5_groupby_viz_filtred[colonne].median()
    mean = house_5_groupby_viz_filtred[colonne].mean()

    axs[i].text(0.95, 0.95, f'Mean: {mean:.2f}\nMedian: {median:.2f}',
transform=axs[i].transAxes, fontsize=10,
                va='top', ha='right')
    axs[i].legend(loc='upper left')
    axs[i].set_title(f"Distribution of {colonne}")
    axs[i].grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_histplot_global.png")
plt.tight_layout()
plt.show()

### D - Fonction de densité

# Fonction de densité de la consommation d'énergie : Foyer 1
house_1_new_numeric_preprocessed["active_power"].plot(title=" Densité
de la consommation d'énergie : Foyer 1", kind="kde")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_density_global.png")
plt.legend()
plt.grid(True)
plt.show()

# Fonction de densité de la consommation d'énergie : Foyer 2
house_2_new_numeric_preprocessed["active_power"].plot(title=" Densité
de la consommation d'énergie : Foyer 2", kind="kde")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_density_global.png")
plt.legend()
plt.grid(True)
```

```
plt.show()

# Fonction de densité de la consommation d'énergie : Foyer 5
house_5_new_numeric_preprocessed["active_power"].plot(title=" Densité
de la consommation d'énergie : Foyer 5", kind="kde")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_density_global.png")
plt.legend()
plt.grid(True)
plt.show()

## **VII - Visualisation des données : Période Spécifique**

# Fonction pour filtrer les données à la date indiquée. ( Date de réfé-
rence ~ 20-04-2014 )

def fct_data_groupby(dataframe, start_date_str, end_date_str, freq_tar-
get="H"):

    '''
    description:
    ------------
    Cette fonction a deux fonctions. D'une part, elle permet de filtred
les données selon une période de temps définie.
    D'autre part, elle sert à grouper les données selon un critère de
périodicité défini.

    argument:
    ---------
    dataframe : La base de données contenant les variables de type da-
tetime(s) et variables explicatives.
    start_date_str : La date du début de filtre  -- format='%Y-%m-%d
%H:%M:%S.%f'.
    end_date_str : Date de fin du filtre -- format='%Y-%m-%d
%H:%M:%S.%f'.
    period_target : Périodicité utilisée pour filtrer les données
freq=["YE", "YS", "ME", "MS", "W", "D", "H", "min", "s"].

    '''

    # Création de la liste destinée à filtrer le dataframe

    List_freq = ["YE", "YS", "ME", "MS", "W", "D", "H", "min", "s"]
    if freq_target in List_freq[:2]:
        List_col_filtred = ["unix_date_filtred", "unix_year", "ac-
tive_power", "apparent_power", "main_rms_voltage"]
    elif freq_target in List_freq[2:4]:
```

```
        List_col_filtred = ["unix_date_filtred", "unix_month", "ac-
tive_power", "apparent_power", "main_rms_voltage"]
    elif freq_target in List_freq[4:6]:
        List_col_filtred = ["unix_date_filtred", "unix_day", "ac-
tive_power", "apparent_power", "main_rms_voltage"]
    elif freq_target in List_freq[6]:
        List_col_filtred = ["unix_date_filtred", "unix_hour", "ac-
tive_power", "apparent_power", "main_rms_voltage"]
    elif freq_target in List_freq[7]:
        List_col_filtred = ["unix_date_filtred", "unix_second", "ac-
tive_power", "apparent_power", "main_rms_voltage"]

     # creation du dataframe filtré en fonction de la période de filtre

    df = pd.DataFrame(dataframe)
    start_date =  pd.to_datetime(start_date_str)
    end_date = pd.to_datetime(end_date_str)
    df_filtred = df[
        (df.index >= start_date) &
        (df.index < end_date)
    ]
    df_filtred["unix_date_filtred"] = df_filtred.index.floor(freq_tar-
get)
    df_filtred_freq = df_filtred[List_col_filtred]


     # Création du dataframe groupé en fonction de la périodicité de
filtre
    df_fitred_groupby = df_filtred_freq.groupby("unix_date_fil-
tred").agg({
        "active_power": "mean",
        "apparent_power": "mean",
        "main_rms_voltage": "mean"

    }).reindex() # set_index("unix_date_filtred")

    #df_filtred_groupby_reindex = df_fitred_groupby["unix_date_fil-
tred"]

    return df_filtred_freq, df_fitred_groupby



# Création de dataframe pour la visualisation des données avec une pé-
riode spécifique
house_1_filtred_freq, house_1_groupby = fct_data_groupby(data-
frame=house_1_new_numeric_preprocessed, start_date_str="2013-09-17
00:00:00", end_date_str="2013-09-18 00:00:00", freq_target="H")
```

```
house_2_filtred_freq, house_2_groupby = fct_data_groupby(data-
frame=house_2_new_numeric_preprocessed, start_date_str="2013-09-17
00:00:00", end_date_str="2013-09-18 00:00:00", freq_target="H")
house_5_filtred_freq, house_5_groupby = fct_data_groupby(data-
frame=house_5_new_numeric_preprocessed, start_date_str="2014-09-17
00:00:00", end_date_str="2014-09-18 00:00:00", freq_target="H")

### A - Foyer 1

# Base de données Filtrée en fonction d'une journée : Foyer 1
house_1_filtred_freq.head(5)

# Base de données Groupée en fonction d'une journée : Foyer 1
house_1_groupby.head(5)

### B - Foyer 2

# Base de données Filtrée en fonction d'une journée : Foyer 2
house_2_filtred_freq.head(5)

# Base de données Groupée en fonction d'une journée : Foyer 2
house_2_groupby.head(5)

### C - Foyer 5

# Base de données Filtrée en fonction d'une journée : Foyer 5
house_5_filtred_freq.head(5)

 # # Base de données Groupée en fonction d'une journée : Foyer 5
 house_5_groupby.head(5)

# Sauvegarder les dataset house_i_filtred_freq
house_1_filtred_freq.to_csv("/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_1_filtred_freq.csv", in-
dex=True)
house_2_filtred_freq.to_csv("/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_2_filtred_freq.csv", in-
dex=True)
house_5_filtred_freq.to_csv("/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_5_filtred_freq.csv", in-
dex=True)

# Sauvegarder les dataset house_i_groupby

house_1_groupby.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataM-
ining__/data_preprocessed/house_1_groupby.csv", index=True)
house_2_groupby.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataM-
ining__/data_preprocessed/house_2_groupby.csv", index=True)
```

```
house_5_groupby.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataM-
ining__/data_preprocessed/house_5_groupby.csv", index=True)

## **VIII - Visualisation des données : Période Spécifique**

### A - Plot

# Consommation d'énergie du 17-09-2013 : Foyer 1
house_1_filtred_freq[["apparent_power", "active_power", "main_rms_vol-
tage"]].plot(title="Consommation d'énergie du Foyer 1")
plt.xlabel("Date : 17-09-2013")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_energy_consumption_specific.png")


# Consommation d'énergie du 17-09-2013 : Foyer 2
house_2_filtred_freq[["apparent_power","active_power", "main_rms_vol-
tage"]].plot(title="Consommation d'énergie du Foyer 2")
plt.xlabel("Date : 17-09-2013")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_energy_consumption_specific.png")


# Consommation d'énergie du 17-09-2014 : Foyer 5
house_2_filtred_freq[[ "apparent_power", "active_power", "main_rms_vol-
tage"]].plot(title="Consommation d'énergie du Foyer 5")
plt.xlabel("Date : 17-09-2014")
plt.ylabel("W")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_energy_consumption_specific.png")


### B - Boxplot


# Boxplot Consommation d'énergie du 17-09-2013 : Foyer 1
plt.figure(figsize=(12,6))
sns.boxplot(data=house_1_filtred_freq,
            x="unix_hour",
            y="active_power")
```

```
plt.title(f"Consommation d'énergie : Foyer 1")
plt.xlabel("Date : 17-09-2013")
plt.ylabel("W")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_boxplot_specific.png")
plt.grid(True)
plt.show()


# Boxplot Consommation d'énergie du 17-09-2013 : Foyer 2
plt.figure(figsize=(12,6))
sns.boxplot(data=house_2_filtred_freq,
            x="unix_hour",
            y="active_power")

plt.title(f"Consommation d'énergie : Foyer 2")
plt.xlabel("Date : 17-09-2013")
plt.ylabel("W")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_boxplot_specific.png")
plt.grid(True)
plt.show()


# Boxplot Consommation d'énergie du 17-09-2014 : Foyer 5
plt.figure(figsize=(12,6))
sns.boxplot(data=house_5_filtred_freq,
            x="unix_hour",
            y="active_power")

plt.title(f"Consommation d'énergie : Foyer 5")
plt.xlabel("Date : 17-09-2014 ")
plt.ylabel("W")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_boxplot_specific.png")
plt.grid(True)
plt.show()


### C - Histogramme


# Histogramme de la consommation d'énergie du foyer 1
house_1_filtred_hist_specific = house_1_filtred_freq[["active_power",
"apparent_power", "main_rms_voltage"]]

fig, axs = plt.subplots(1, 3, figsize=(12, 6))
axs = axs.ravel()
```

```
for i, colonne in enumerate(house_1_filtred_hist_specific.se-
lect_dtypes(include=["float", "integer"]).columns):
    sns.distplot(house_1_filtred_hist_specific[colonne], ax=axs[i],
bins='auto', hist=True, rug=False, kde=False,
                 color='#2E6A9D')
    axs[i].axvline(house_1_filtred_hist_specific[colonne].mean(),
color='red', linestyle='--', label='mean')
    axs[i].axvline(house_1_filtred_hist_specific[colonne].median(),
color='green', linestyle='-.', label='median')

    median = house_1_filtred_hist_specific[colonne].median()
    mean = house_1_filtred_hist_specific[colonne].mean()

    axs[i].text(0.95, 0.95, f'Mean: {mean:.2f}\nMedian: {median:.2f}',
transform=axs[i].transAxes, fontsize=10,
                va='top', ha='right')
    axs[i].legend(loc='upper left')
    axs[i].set_title(f"Distribution of {colonne}")
    axs[i].grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_histplot_specific.png")
plt.tight_layout()

plt.show()


# Histogramme de la consommation d'énergie du foyer 1
house_2_filtred_hist_specific = house_2_filtred_freq[["active_power",
"apparent_power", "main_rms_voltage"]]

fig, axs = plt.subplots(1, 3, figsize=(12, 6))
axs = axs.ravel()

for i, colonne in enumerate(house_2_filtred_hist_specific.se-
lect_dtypes(include=["float", "integer"]).columns):
    sns.distplot(house_2_filtred_hist_specific[colonne], ax=axs[i],
bins='auto', hist=True, rug=False, kde=False,
                 color='#2E6A9D')
    axs[i].axvline(house_2_filtred_hist_specific[colonne].mean(),
color='red', linestyle='--', label='mean')
    axs[i].axvline(house_2_filtred_hist_specific[colonne].median(),
color='green', linestyle='-.', label='median')

    median = house_2_filtred_hist_specific[colonne].median()
    mean = house_2_filtred_hist_specific[colonne].mean()

    axs[i].text(0.95, 0.95, f'Mean: {mean:.2f}\nMedian: {median:.2f}',
transform=axs[i].transAxes, fontsize=10,
```

```
                va='top', ha='right')
    axs[i].legend(loc='upper left')
    axs[i].set_title(f"Distribution of {colonne}")
    axs[i].grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_histplot_specific.png")
plt.tight_layout()

plt.show()


# Histogramme de la consommation d'énergie du foyer 5
house_5_filtred_hist_specific = house_5_filtred_freq[["active_power",
"apparent_power", "main_rms_voltage"]]

fig, axs = plt.subplots(1, 3, figsize=(12, 6))
axs = axs.ravel()

for i, colonne in enumerate(house_5_filtred_hist_specific.se-
lect_dtypes(include=["float", "integer"]).columns):
    sns.distplot(house_5_filtred_hist_specific[colonne], ax=axs[i],
bins='auto', hist=True, rug=False, kde=False,
                 color='#2E6A9D')
    axs[i].axvline(house_5_filtred_hist_specific[colonne].mean(),
color='red', linestyle='--', label='mean')
    axs[i].axvline(house_5_filtred_hist_specific[colonne].median(),
color='green', linestyle='-.', label='median')

    median = house_5_filtred_hist_specific[colonne].median()
    mean = house_5_filtred_hist_specific[colonne].mean()

    axs[i].text(0.95, 0.95, f'Mean: {mean:.2f}\nMedian: {median:.2f}',
transform=axs[i].transAxes, fontsize=10,
                va='top', ha='right')
    axs[i].legend(loc='upper left')
    axs[i].set_title(f"Distribution of {colonne}")
    axs[i].grid(True)
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_histplot_specific.png")
plt.tight_layout()

plt.show()


### D - Fonction de densité


# Fonction de densité de la consommation d'énergie : Foyer 1
```

```
house_1_filtred_freq["active_power"].plot(title=" Densité de la consom-
mation d'énergie : Foyer 1", kind="kde")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_1_density_global.png")
plt.legend()
plt.show()

# Fonction de densité de la consommation d'énergie : Foyer 2
house_2_filtred_freq["active_power"].plot(title=" Densité de la consom-
mation d'énergie: Foyer 2", kind="kde")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_2_density_global.png")
plt.legend()
plt.show()

# Fonction de densité de la consommation d'énergie : Foyer 5
house_5_filtred_freq["active_power"].plot(title=" Densité de la consom-
mation d'énergie: Foyer 5", kind="kde")
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/house_5_density_global.png")
plt.legend()
plt.show()

### D - PCOLOR MESH IMAGE


### Création d'un dataframe destiné à groupé les house_i_grouby
var_target = "active_power"
data_pcolor = pd.DataFrame({
    "unix_index": house_1_groupby.index.hour,
    "house1": house_1_groupby[var_target].tolist(),
    "house2": house_2_groupby[var_target].tolist(),
    "house5": house_5_groupby[var_target].tolist(),
})
data_pcolor


# Sauvegarder le dataframe
data_pcolor.to_csv("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/data_preprocessed/data_pcolor.csv", index=False)


from matplotlib.colors import LinearSegmentedColormap, Normalize

# Préparation des données pour pcolormesh
values = data_pcolor[["house1", "house2", "house5"]].T.values  # Trans-
pose pour avoir les maisons en ordonnées
hours = data_pcolor["unix_index"].values  # Heures
houses = ["House 1", "House 2", "House 5"]  # Noms des maisons
```

```
# Normalisation des valeurs
norm = Normalize(vmin=np.min(values), vmax=np.max(values))

# Création du graphique
fig, ax = plt.subplots(figsize=(16, 6))
c = ax.pcolormesh(hours, range(len(houses)), values, norm=norm,
edgecolors='w', linewidth=0.5, cmap="Blues") # https://mat-
plotlib.org/stable/users/explain/colors/colormaps.html

# Configuration des axes
ax.set_xticks(hours + 0.5)  # Centrer les ticks
ax.set_xticklabels(hours, rotation=0)
ax.set_yticks(np.arange(len(houses)) + 0.5)  # Centrer les ticks
ax.set_yticklabels(houses)
ax.set_xlabel("Date")
ax.set_ylabel("Maisons")
ax.set_title("Consommation d'énergie par maison et par heure")
ax.set_aspect('equal')

# Ajout de la barre de couleur avec ajustement
cbar = fig.colorbar(c, ax=ax, orientation='vertical', fraction=0.03,
pad=0.04, shrink=0.8, label="Consommation (kWh)")

# Affichage
plt.tight_layout()
plt.savefig("/content/drive/MyDrive/ColabNotebooks/1__DataMin-
ing__/graphics/pcolormesh_specific.png")
plt.show()
```

7.3 PROGRAMME : MODELE DE SERIES TEMPORELLES : ARIMA

```
## **I - Chemin vers le Google Drive**
# Code pour accéder aux dossiers Google Drive
from google.colab import drive
drive.mount('/content/drive')

## **II - Importation des librairies**

!pip uninstall -y numpy pmdarima
!pip install numpy==1.24.4  # Version stable compatible avec pmdarima
!pip install --no-cache-dir pmdarima

# Runtime > Restart runtime (Redémarrer le runtime)
import pmdarima as pm

# Importer les librairies
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os
import statsmodels
import tabulate

# Importation à partir de statsmodels
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA

## **III - Paramétrages des librairies**

# Paramètres de Pandas

pd.set_option("display.max.rows", 500)
pd.set_option("display.max.columns", 500)
pd.set_option("display.width", 1000)

# Paramètres de os
print(os.getcwd())
os.makedirs("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics", exist_ok=True)
os.makedirs("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables", exist_ok=True) # Version
html ou Latex

## **IV - Chargement des bases de données**
```

```
# Charger de la base de données à l'aide de pandas
house_1_df = pd.read_csv(r"/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_1_new_numeric_prepro-
cessed.csv")
house_2_df = pd.read_csv(r"/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_2_new_numeric_prepro-
cessed.csv")
house_5_df = pd.read_csv(r"/content/drive/MyDrive/ColabNote-
books/1__DataMining__/data_preprocessed/house_5_new_numeric_prepro-
cessed.csv")

# head.dataframe de house_1_new_numeric_preprocessed (house_1_df) :
Foyer 1
house_1_df.head(5)

# head.dataframe de house_2_new_numeric_preprocessed (house_2_df) :
Foyer 2
house_2_df.head(5)

# head.dataframe de house_5_new_numeric_preprocessed (house_5_df) :
Foyer 5
house_5_df.head(5)

## **V - Analyse des bases de données**
### 1 - Vérifier le type des variables

# Vérifier le type de l'index house_1_df : Foyer 1
print(house_1_df.index.dtype)
print("-------------x---------------")
print(house_1_df.info())

# Vérifier le type de l'index house_1_df : Foyer 2
print(house_2_df.index.dtype)
print("-------------x---------------")
print(house_2_df.info())

# Vérifier le type de l'index house_1_df : Foyer 5
print(house_5_df.index.dtype)
print("------------- x---------------")
print(house_5_df.info())

### 2 - Convertir l'index en datetime
# Convertir le type de l'index en datetime
house_1_df.index = pd.to_datetime(house_1_df["unix"])
house_2_df.index = pd.to_datetime(house_2_df["unix"])
house_5_df.index = pd.to_datetime(house_5_df["unix"])

# Vérifier à nouveau le type de l'index
```

```
print(house_1_df.index.dtype)
print("-------------x---------------")
print(house_2_df.index.dtype)
print("-------------x---------------")
print(house_5_df.index.dtype)

### 3 - Traitement des valeurs nulles et des valeurs **manquantes**
# Vérifier si house_1_new contient des valeurs manquantes : Foyer 1
print(house_1_df.isna().sum())
print("----------x------------")
# Vérifier si house_1_new contient des valeurs nulles : Foyer 1
print(house_1_df.isnull().sum())

# Vérifier si house_1_new contient des valeurs manquantes : Foyer 2
print(house_2_df.isna().sum())
print("----------x------------")
# Vérifier si house_1_new contient des valeurs nulles
print(house_2_df.isnull().sum())

# Vérifier si house_1_new contient des valeurs manquantes : Foyer 5
print(house_5_df.isna().sum())
print("----------x------------")
# Vérifier si house_1_new contient des valeurs nulles
print(house_5_df.isnull().sum())

# Supprimer les valeurs manquantes
control_drop = False
if control_drop == True :
    house_1_df.dropna(inplace=True)
    house_2_df.dropna(inplace=True)
    house_5_df.dropna(inplace=True)

#### 4 - Base de données groupées en Heure
# Fonction pour filtrer les données à la date indiquée. ( Date de réfé-
rence ~ 20-04-2014 )

def fct_data_groupby_timeserie(dataframe, freq_target="H"):

    '''
    description:
    ------------
    Cette fonction permet de grouper les données selon un critère de
périodicité défini.

    argument:
    ---------
    dataframe : La base de données contenant les variables de type da-
tetime(s) et variables explicatives.
```

```
    period_target : Périodicité utilisée pour filtrer les données
freq=["YE", "YS", "ME", "MS", "W", "D", "H", "min", "s"].

    '''
    # Création de la variable (floor) utilisée pour grouper les données
    df = pd.DataFrame(dataframe)
    df["unix_id_group"] = df.index.floor(freq_target)

     # Création du dataframe groupé en fonction de la périodicité de
filtre
    df_groupby = df.groupby("unix_id_group").agg({
        "active_power": "mean",
        "apparent_power": "mean",
        "main_rms_voltage": "mean"

    }).reindex()

    df_groupby.reindex(df["unix_id_group"].tolist())
    return df_groupby

# Création des dataframe house_i_df_ts
house_1_df_ts = fct_data_groupby_timeserie(dataframe=house_1_df,
freq_target="H")
house_2_df_ts = fct_data_groupby_timeserie(dataframe=house_2_df,
freq_target="H")
house_5_df_ts = fct_data_groupby_timeserie(dataframe=house_5_df,
freq_target="H")

# head.dataframe pour la base de données house_2_df_ts: Foyer 2
house_1_df_ts.head(5)

# head.dataframe pour la base de données house_1_df_ts: Foyer 1
house_2_df_ts.head(5)

# head.dataframe pour la base de données house_5_df_ts: Foyer 5
house_5_df_ts.head(5)

# Sauvegarder les fichiers house_i_df_ts
house_1_df_ts.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_1_df_ts.csv", in-
dex=True)
house_2_df_ts.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_2_df_ts.csv", in-
dex=True)
house_5_df_ts.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_5_df_ts.csv", in-
dex=True)
```

```
## **VI - Méthodologie de BOX-JENKINS**
#### 1.A - Graphique de la consommation d'énergie par foyer

# Graphique de la consommation d'énergie globale au sein des foyers

from IPython.display import Image, display

display(Image(filename=r"/content/drive/MyDrive/ColabNote-
books/1__DataMining__/graphics/house_1_energy_consumption_global.png"))
display(Image(filename=r"/content/drive/MyDrive/ColabNote-
books/1__DataMining__/graphics/house_2_energy_consumption_global.png"))
display(Image(filename=r"/content/drive/MyDrive/ColabNote-
books/1__DataMining__/graphics/house_5_energy_consumption_global.png"))

#### 1.B - Graphiques de la fonction d'autocorrelation et la fonction
d'autocorrelation partielle
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import numpy as np
# Fonction d'autocorrelation et Fonction d'autocorrelation partielle :
Foyer 1

# Créer les subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 6))

# Tracer l'ACF
plot_acf(house_1_df_ts['active_power'], lags=30, zero=True, ax=ax1)
ax1.set_title("ACF - Consommation d'énergie (W) : Foyer 1")
ax1.set_xlabel('Lag')
ax1.set_ylabel('Corrélation')
ax1.grid(True)

# Ajuster les graduations sur l'axe x pour l'ACF
ax1.set_xticks(np.arange(0, 31, 1))

# Tracer le PACF
plot_pacf(house_1_df_ts['active_power'], lags=30, zero=True, ax=ax2)
ax2.set_title("PACF - Consommation d'énergie (W) : Foyer 1")
ax2.set_xlabel('Lag')
ax2.set_ylabel('Corrélation partielle')
ax2.grid(True)

# Ajuster les graduations sur l'axe x pour le PACF
ax2.set_xticks(np.arange(0, 31, 1))

# Ajuster les subplots
plt.tight_layout()
```

```
# Sauvegarder le fichier
#plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_1_caf_pcaf.png")

# Afficher le graphique
plt.show()

# Fonction d'autocorrelation et Fonction d'autocorrelation partielle :
Foyer 2

# Créer les subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 6))

# Tracer l'ACF
plot_acf(house_2_df_ts['active_power'], lags=30, zero=True, ax=ax1)
ax1.set_title("ACF - Consommation d'énergie (W) : Foyer 2")
ax1.set_xlabel('Lag')
ax1.set_ylabel('Corrélation')
ax1.grid(True)

# Ajuster les graduations sur l'axe x pour l'ACF
ax1.set_xticks(np.arange(0, 31, 1))

# Tracer le PACF
plot_pacf(house_2_df_ts['active_power'], lags=30, zero=True, ax=ax2)
ax2.set_title("PACF - Consommation d'énergie (kWh) : Foyer 2")
ax2.set_xlabel('Lag')
ax2.set_ylabel('Corrélation partielle')
ax2.grid(True)

# Ajuster les graduations sur l'axe x pour le PACF
ax2.set_xticks(np.arange(0, 31, 1))

# Ajuster les subplots
plt.tight_layout()

# Sauvegarder le fichier
#plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_2_caf_pcaf.png")

# Afficher le graphique
plt.show()

# Fonction d'autocorrelation et Fonction d'autocorrelation partielle :
Foyer 5

# Créer les subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 6))
```

```
# Tracer l'ACF
plot_acf(house_5_df_ts['active_power'], lags=30, zero=True, ax=ax1)
ax1.set_title("ACF - Consommation d'énergie (W) : Foyer 5")
ax1.set_xlabel('Lag')
ax1.set_ylabel('Corrélation')
ax1.grid(True)

# Ajuster les graduations sur l'axe x pour l'ACF
ax1.set_xticks(np.arange(0, 31, 1))

# Tracer le PACF
plot_pacf(house_5_df_ts['active_power'], lags=30, zero=True, ax=ax2)
ax2.set_title("PACF - Consommation d'énergie (kWh) : Foyer 5")
ax2.set_xlabel('Lag')
ax2.set_ylabel('Corrélation partielle')
ax2.grid(True)

# Ajuster les graduations sur l'axe x pour le PACF
ax2.set_xticks(np.arange(0, 31, 1))

# Ajuster les subplots
plt.tight_layout()

# Sauvegarder le fichier
#plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_5_caf_pcaf.png")

# Afficher le graphique
plt.show()

from tabulate import tabulate

# Test de Dickey-Fuller : Foyer 1
house_1_adfuller = adfuller(house_1_df_ts["active_power"])
house_1_dickeyfuller = [
    ["Valeur de test", house_1_adfuller[0]],
    ["P-valeur", house_1_adfuller[1]],
    ["Conclusion", "La série est stationnare" if house_1_adfuller[1] <
0.05 else "La série est non stationnaire"]
]

# Afficher les résultats sous la forme d'un dataframe
house_1_dickeyfuller

# Test de Dickey-Fuller : Foyer 2

house_2_adfuller = adfuller(house_2_df_ts["active_power"])
```

```
house_2_dickeyfuller = [
    ["Valeur de test", house_2_adfuller[0]],
    ["P-valeur", house_2_adfuller[1]],
    ["Conclusion", "La série est stationnare" if house_2_adfuller[1] <
0.05 else "La série est non stationnaire"]
]

# Afficher les résultats sous la forme d'un dataframe
house_2_dickeyfuller

# Test de Dickey-Fuller : Foyer 1

house_5_adfuller = adfuller(house_5_df_ts["active_power"])
house_5_dickeyfuller = [
    ["Valeur de test", house_5_adfuller[0]],
    ["P-valeur", house_5_adfuller[1]],
    ["Conclusion", "La série est stationnare" if house_5_adfuller[1] <
0.05 else "La série est non stationnaire"]
]

# Afficher les résultats sous la forme d'un dataframe
house_5_dickeyfuller

#### 1.D - Graphique de Décomposition
# Effectuer la décomposition saisonnière : Foyer 1
decomposition = seasonal_decompose(x=house_1_df_ts['active_power'],
model='additive', period=12)

# Extraire les composantes de la décomposition
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Afficher les composantes de la décomposition
plt.figure(figsize=(12, 8))

plt.subplot(411)
plt.plot(house_1_df_ts['active_power'], label='Série originale')
plt.legend(loc='best')

plt.subplot(412)
plt.plot(trend, label='Tendance')
plt.legend(loc='best')

plt.subplot(413)
plt.plot(seasonal, label='Saisonnalité')
plt.legend(loc='best')
```

```
plt.subplot(414)
plt.plot(residual, label='Résidus')
plt.legend(loc='best')
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_1_decomposi-
tion.png")

plt.tight_layout()
plt.show()

# Effectuer la décomposition saisonnière : Foyer 2
decomposition = seasonal_decompose(x=house_2_df_ts['active_power'],
model='additive', period=12)

# Extraire les composantes de la décomposition
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Afficher les composantes de la décomposition
plt.figure(figsize=(12, 8))

plt.subplot(411)
plt.plot(house_2_df_ts['active_power'], label='Série originale')
plt.legend(loc='best')

plt.subplot(412)
plt.plot(trend, label='Tendance')
plt.legend(loc='best')

plt.subplot(413)
plt.plot(seasonal, label='Saisonnalité')
plt.legend(loc='best')

plt.subplot(414)
plt.plot(residual, label='Résidus')
plt.legend(loc='best')
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_2_decomposi-
tion.png")

plt.tight_layout()
plt.show()

# Effectuer la décomposition saisonnière : Foyer 5
decomposition = seasonal_decompose(x=house_5_df_ts['active_power'],
model='additive', period=12)
```

```
# Extraire les composantes de la décomposition
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Afficher les composantes de la décomposition
plt.figure(figsize=(12, 8))

plt.subplot(411)
plt.plot(house_5_df_ts['active_power'], label='Série originale')
plt.legend(loc='best')

plt.subplot(412)
plt.plot(trend, label='Tendance')
plt.legend(loc='best')

plt.subplot(413)
plt.plot(seasonal, label='Saisonnalité')
plt.legend(loc='best')

plt.subplot(414)
plt.plot(residual, label='Résidus')
plt.legend(loc='best')
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_5_decomposi-
tion.png")

plt.tight_layout()
plt.show()

### 2 - Différenciation
# Différenciation pour rendre la série stationnaire : Foyer 1
differenced_1 = house_1_df_ts['active_power'].diff().dropna()

# Afficher la série différienciée
plt.figure(figsize=(6, 4))
plt.plot(differenced_1)
plt.title("Série temporelle différienciée (Consommation d'énergie) :
Foyer 1")
plt.xlabel('Date')
plt.ylabel('Différence')
plt.grid(True)

#plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_1_differencia-
tion.png")
plt.show()
```

```
# Différenciation pour rendre la série stationnaire : Foyer 2
differenced_2 = house_2_df_ts['active_power'].diff().dropna()

# Afficher la série différienciée
plt.figure(figsize=(6,4 )) # 10,6
plt.plot(differenced_2)
plt.title("Série temporelle différienciée (Consommation d'énergie) :
Foyer 2")
plt.xlabel('Date')
plt.ylabel('Différence')
plt.grid(True)
#plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_2_differencia-
tion.png")
plt.show()

# Différenciation pour rendre la série stationnaire : Foyer 5
differenced_5 = house_5_df_ts['active_power'].diff().dropna()

# Afficher la série différienciée
plt.figure(figsize=(10, 4))
plt.plot(differenced_5)
plt.title("Série temporelle différienciée (Consommation d'énergie) :
Foyer 5")
plt.xlabel('Date')
plt.ylabel('Différence')
plt.grid(True)
#plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_5_differencia-
tion.png")
plt.show()

### 3 - Identification des Modèles ARIMA(p,d,q)
identification = pd.DataFrame({
    "House": ["house_1", "house_2", "house_5"],
    "p": [1, 2, 2],
    "d": [0, 1, 1],
    "q": [2, 1, 1]
})

identification
### 4 - Train Test Split ARIMA
#### 4.A - Création des bases de données d'entraînement et des bases de
données test

# Séparer le jeu de données : house_1 en 80% train et 20% test
house_1_train_size = int(len(house_1_df_ts) * 0.8)
house_1_train = house_1_df_ts.iloc[:house_1_train_size]
```

```
house_1_test = house_1_df_ts.iloc[house_1_train_size:]

# Séparer le jeu de données : house_1 en 80% train et 20% test
house_2_train_size = int(len(house_2_df_ts) * 0.8)
house_2_train = house_2_df_ts.iloc[:house_2_train_size]
house_2_test = house_2_df_ts.iloc[house_2_train_size:]

# Séparer le jeu de données : house_1 en 80% train et 20% test
house_5_train_size = int(len(house_5_df_ts) * 0.8)
house_5_train = house_5_df_ts.iloc[:house_5_train_size]
house_5_test = house_5_df_ts.iloc[house_5_train_size:]

#### 4.B - Visualition des bases de données d'entraînement et des bases
de données test

fig, ax = plt.subplots(figsize=(15, 5))
house_1_train["active_power"].plot(ax=ax, label='Training Set', ti-
tle='Data Train/Test Split : Foyer 1')
house_1_test["active_power"].plot(ax=ax, label='Test Set')
ax.axvline(str(house_1_df_ts.iloc[house_1_train_size].name),
color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_1_splitdataset.png"
)
plt.grid(True)
plt.show()

fig, ax = plt.subplots(figsize=(15, 5))
house_2_train["active_power"].plot(ax=ax, label='Training Set', ti-
tle='Data Train/Test Split : Foyer 2')
house_2_test["active_power"].plot(ax=ax, label='Test Set')
ax.axvline(str(house_2_df_ts.iloc[house_2_train_size].name),
color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_2_splitdataset.png"
)
plt.grid(True)
plt.show()

fig, ax = plt.subplots(figsize=(15, 5))
house_5_train["active_power"].plot(ax=ax, label='Training Set', ti-
tle='Data Train/Test Split : Foyer 5')
house_5_test["active_power"].plot(ax=ax, label='Test Set')
ax.axvline(str(house_5_df_ts.iloc[house_5_train_size].name),
color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
```

```
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_5_splitdataset.png"
)
plt.grid(True)
plt.show()

### 5 - Estimation des modèles ARIMA(p,d,q)

#### 5.A - Tableau Statsmodels

# Importer la librairie ARIMA  à partir de statsmodels (Rappel)
from statsmodels.tsa.arima.model import ARIMA

house_1_model = ARIMA(house_1_train["active_power"], order=identifica-
tion.iloc[0,1:].tolist())

# Ajuster le modèle
house_1_model_fit = house_1_model.fit()

# Afficher le résumé du modèle
print(house_1_model_fit.summary())

house_2_model = ARIMA(house_2_train["active_power"], order=identifica-
tion.iloc[1, 1:].tolist())

# Ajuster le modèle
house_2_model_fit = house_2_model.fit()

# Afficher le résumé du modèle
print(house_2_model_fit.summary())

house_5_model = ARIMA(house_5_train["active_power"], order=identifica-
tion.iloc[2, 1:].tolist())

# Ajuster le modèle
house_5_model_fit = house_5_model.fit()

# Afficher le résumé du modèle
print(house_5_model_fit.summary())

### 6 - Prévision avec les modèles ARIMA(p,d,q)
#### 6.A - Foyer 1

# Prévision de la consommation d'énergie - Base de données d'entraîne-
ment
house_1_train_prediction = house_1_model_fit.pre-
dict(start=house_1_train.index[0], end=house_1_train.index[-1])
```

```
# Prévision de la consommation d'énergie - Base de données test
house_1_test_prediction = house_1_model_fit.fore-
cast(steps=len(house_1_test))

# Mettre les données prédites de la base de données d'entraînement sous
format dataframe
house_1_train_prediction_df = pd.DataFrame(data=house_1_train_predic-
tion.values , index=house_1_train.index, columns=["active_power"])

# Mettre les données prédites de la base de données de test sous format
dataframe
house_1_test_prediction_df = pd.DataFrame(data=house_1_test_predic-
tion.values , index=house_1_test.index, columns=["active_power"])

# Sauvegarder des données prédites de la base de données d'entraînement
sous format csv
house_1_train_prediction_df.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_1_train_predic-
tion_df.csv", index=True)

# Sauvegarder des données prédites de la base de données de tes sous
format csv
house_1_test_prediction_df.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_1_test_pre-
dicted_df.csv", index=True)

# Tracer les prédictions et les données réelles avec les courbes col-
lées
plt.figure(figsize=(10, 6))

# Données d'entraînement en bleu
plt.plot(house_1_train["active_power"].index, house_1_train["active_po-
wer"].values, label="Ensemble d'entraînement", color="blue")

# Données de test en orange
plt.plot(house_1_test["active_power"].index, house_1_test["ac-
tive_power"].values, label="Ensemble de test (réel)", color="orange",
linestyle="--")

# Données prédites en rouge
plt.plot(house_1_test_prediction_df["active_power"].index,
house_1_test_prediction.values, label="Prédictions (Test)",
color="red")

# Ajouter les labels et le titre
plt.xlabel("Date")
plt.ylabel("Consommation d'énergie (hWh) : Foyer 1")
plt.title("Prédictions du modèle ARIMA(2, 1, 1) : Foyer 1 ")
```

```
# ax.axvline('2014-07-06  00:00:00', color='black', ls='--') --------->
Vérifier après.
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_1_energy_forecast-
ing.png")
plt.show()

from sklearn.metrics import mean_absolute_error, mean_absolute_percent-
age_error, root_mean_squared_error

# Mesures de performance sur l'ensemble d'entraînement
house_1_train_mae = mean_absolute_error(house_1_train["active_power"],
house_1_train_prediction_df["active_power"])
house_1_train_mape = mean_absolute_percentage_error(house_1_train["ac-
tive_power"], house_1_train_prediction_df["active_power"])
house_1_train_rmse = root_mean_squared_error(house_1_train["ac-
tive_power"], house_1_train_prediction_df["active_power"])

# Mesures de performance sur l'ensemble de test
house_1_test_mae = mean_absolute_error(house_1_test["active_power"],
house_1_test_prediction_df["active_power"])
house_1_test_mape = mean_absolute_percentage_error(house_1_test["ac-
tive_power"], house_1_test_prediction_df["active_power"])
house_1_test_rmse = root_mean_squared_error(house_1_test["ac-
tive_power"], house_1_test_prediction_df["active_power"])


# Créer un DataFrame pour afficher les mesures de performance
house_1_performance_df = pd.DataFrame({
    "Métrique": ['MAE', 'MAPE', 'RMSE'],
    "Train dataset": [house_1_train_mae, house_1_train_mape,
house_1_train_rmse],
    "Test dataset": [house_1_test_mae, house_1_test_mape,
house_1_test_rmse]
})
house_1_performance_df

#### 6.B - Foyer 2
#  Prévision de la consommation d'énergie - Base de données d'entraîne-
ment
house_2_train_prediction = house_2_model_fit.pre-
dict(start=house_2_train.index[0], end=house_2_train.index[-1])

# Prévision de la consommation d'énergie - Base de données test
house_2_test_prediction = house_2_model_fit.fore-
cast(steps=len(house_2_test))
```

```
# Mettre les données prédites de la base de données d'entraînement sous
format dataframe
house_2_train_prediction_df = pd.DataFrame(data=house_2_train_predic-
tion.values , index=house_2_train.index, columns=["active_power"])

# Mettre les données prédites de la base de données de test sous format
dataframe
house_2_test_prediction_df = pd.DataFrame(data=house_2_test_predic-
tion.values , index=house_2_test.index, columns=["active_power"])

# Sauvegarder des données prédites de la base de données d'entraînement
sous format csv
house_2_train_prediction_df.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_2_train_predic-
tion_df.csv", index=True)

# Sauvegarder des données prédites de la base de données de tes sous
format csv
house_2_test_prediction_df.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_2_test_predic-
tion_df.csv", index=True)

# Tracer les prédictions et les données réelles avec les courbes col-
lées
plt.figure(figsize=(10, 6))

# Données d'entraînement en bleu
plt.plot(house_2_train["active_power"].index, house_2_train["active_po-
wer"].values, label="Ensemble d'entraînement", color="blue")

# Données de test en orange
plt.plot(house_2_test["active_power"].index, house_2_test["ac-
tive_power"].values, label="Ensemble de test (réel)", color="orange",
linestyle="--")

# Données prédites en rouge
plt.plot(house_2_test_prediction_df["active_power"].index,
house_2_test_prediction.values, label="Prédictions (Test)",
color="red")

# Ajouter les labels et le titre
plt.xlabel("Date")
plt.ylabel("Consommation d'énergie (hWh)")
plt.title("Prédictions du modèle ARIMA(2, 1, 1) : Foyer 2 ")
plt.legend()
plt.grid(True)
```

```
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_2_energy_forecast-
ing.png")
plt.show()

from sklearn.metrics import mean_absolute_error, mean_absolute_percent-
age_error, root_mean_squared_error

# Mesures de performance sur l'ensemble d'entraînement
house_2_train_mae = mean_absolute_error(house_2_train["active_power"],
house_2_train_prediction_df["active_power"])
house_2_train_mape = mean_absolute_percentage_error(house_2_train["ac-
tive_power"], house_2_train_prediction_df["active_power"])
house_2_train_rmse = root_mean_squared_error(house_2_train["ac-
tive_power"], house_2_train_prediction_df["active_power"])

# Mesures de performance sur l'ensemble de test
house_2_test_mae = mean_absolute_error(house_2_test["active_power"],
house_2_test_prediction_df["active_power"])
house_2_test_mape = mean_absolute_percentage_error(house_2_test["ac-
tive_power"], house_2_test_prediction_df["active_power"])
house_2_test_rmse = root_mean_squared_error(house_2_test["ac-
tive_power"], house_2_test_prediction_df["active_power"])


# Créer un DataFrame pour afficher les mesures de performance
house_2_performance_df = pd.DataFrame({
    "Métrique": ['MAE', 'MAPE', 'RMSE'],
    "Train dataset": [house_2_train_mae, house_2_train_mape,
house_2_train_rmse],
    "Test dataset": [house_2_test_mae, house_2_test_mape,
house_2_test_rmse]
})
house_2_performance_df

#### 6.C - Foyer 5

#  Prévision de la consommation d'énergie - Base de données d'entraîne-
ment
house_5_train_prediction = house_5_model_fit.pre-
dict(start=house_5_train.index[0], end=house_5_train.index[-1])

# Prévision de la consommation d'énergie - Base de données test
house_5_test_prediction = house_5_model_fit.fore-
cast(steps=len(house_5_test))

# Mettre les données prédites de la base de données d'entraînement sous
format dataframe
```

```
house_5_train_prediction_df = pd.DataFrame(data=house_5_train_predic-
tion.values , index=house_5_train.index, columns=["active_power"])

# Mettre les données prédites de la base de données de test sous format
dataframe
house_5_test_prediction_df = pd.DataFrame(data=house_5_test_predic-
tion.values , index=house_5_test.index, columns=["active_power"])

# Sauvegarder des données prédites de la base de données d'entraînement
sous format csv
house_5_train_prediction_df.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_5_train_predic-
tion_df.csv", index=True)

# Sauvegarder des données prédites de la base de données de tes sous
format csv
house_5_test_prediction_df.to_csv("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesTables/house_5_test_predic-
tion_df.csv", index=True)

# Tracer les prédictions et les données réelles avec les courbes col-
lées
plt.figure(figsize=(10, 6))

# Données d'entraînement en bleu
plt.plot(house_5_train["active_power"].index, house_5_train["active_po-
wer"].values, label="Ensemble d'entraînement", color="blue")

# Données de test en orange
plt.plot(house_5_test["active_power"].index, house_5_test["ac-
tive_power"].values, label="Ensemble de test (réel)", color="orange",
linestyle="--")

# Données prédites en rouge
plt.plot(house_5_test_prediction_df["active_power"].index,
house_5_test_prediction.values, label="Prédictions (Test)",
color="red")

# Ajouter les labels et le titre
plt.xlabel("Date")
plt.ylabel("Consommation d'énergie (hWh) : Foyer 5")
plt.title("Prédictions du modèle ARIMA(2, 1, 1) : Foyer 5 ")
plt.legend()
plt.grid(True)
plt.savefig("/content/drive/MyDrive/ColabNote-
books/2__TimeSeriesModel__/TimeSeriesGraphics/house_5_energy_forecast-
ing.png")
plt.show()
```

```
from sklearn.metrics import mean_absolute_error, mean_absolute_percent-
age_error, root_mean_squared_error

# Mesures de performance sur l'ensemble d'entraînement
house_5_train_mae = mean_absolute_error(house_5_train["active_power"],
house_5_train_prediction_df["active_power"])
house_5_train_mape = mean_absolute_percentage_error(house_5_train["ac-
tive_power"], house_5_train_prediction_df["active_power"])
house_5_train_rmse = root_mean_squared_error(house_5_train["ac-
tive_power"], house_5_train_prediction_df["active_power"])

# Mesures de performance sur l'ensemble de test
house_5_test_mae = mean_absolute_error(house_5_test["active_power"],
house_5_test_prediction_df["active_power"])
house_5_test_mape = mean_absolute_percentage_error(house_5_test["ac-
tive_power"], house_5_test_prediction_df["active_power"])
house_5_test_rmse = root_mean_squared_error(house_5_test["ac-
tive_power"], house_5_test_prediction_df["active_power"])


# Créer un DataFrame pour afficher les mesures de performance
house_5_performance_df = pd.DataFrame({
    "Métrique": ['MAE', 'MAPE', 'RMSE'],
    "Train dataset": [house_5_train_mae, house_5_train_mape,
house_5_train_rmse],
    "Test dataset": [house_5_test_mae, house_5_test_mape,
house_5_test_rmse]
})
house_5_performance_df

#### 7 - ALGORITHME D'IDENTIFICATION DES ORDRES DU MODELE ARIMA
#### 7.A - Foyer 1
# Utiliser auto_arima pour trouver le meilleur modèle ARIMA
house_1_pmdarima = pm.auto_arima(house_1_train["active_power"])
print(house_1_pmdarima.summary())

#### 7.B - Foyer 2
# Utiliser auto_arima pour trouver le meilleur modèle ARIMA
house_2_pmdarima = pm.auto_arima(house_2_train["active_power"])
print(house_2_pmdarima.summary())

#### 7.C - Foyer 5
# Utiliser auto_arima pour trouver le meilleur modèle ARIMA
house_5_pmdarima = pm.auto_arima(house_5_train["active_power"])
print(house_5_pmdarima.summary())
```

7.4 PROGRAMME : RESEAUX BAYESIEN DYNAMIQUE LINEAIRE

```
### **I - CHEMIN VERS LE GOOGLE DRIVE**
# Paramètre d'installation de Google Drive à
if (file.exists("/usr/local/lib/python3.6/dist-pack-
ages/google/colab/_ipython.py")) {
  install.packages("R.utils")
  library(R.utils)
  library(httr)
  my_check <- function() {return(TRUE)}
  reassignInPackage("is_interactive", pkgName = "httr", my_check)
  options(rlang_interactive=TRUE)
}
library(googledrive)
library(googlesheets4)

# call authentication forcing interactive login and save in cache.
drive_auth(use_oob = TRUE, cache = TRUE)

### **II - DEFINIR L'ESPACE DE TRAVAIL**
# Définir l'environnement de travail et créer les fichier
setwd

### **III - INSTALLER LES LIBRAIRIES**
install.packages("ggplot2")
install.packages("lubridate")
install.packages("shiny")
install.packages("bnlearn")
install.packages("data.table")
install.packages("dbnR")
install.packages("dplyr")

### **IV - IMPORTER LES LIBRAIRIES**
library(ggplot2)
library(lubridate)
library(shiny)
library(bnlearn)
library(data.table)
library(dbnR)
library(dplyr)

### **V - CHARER LES FICHIERS**
#### 5.1 - Charger les données

# Charger le fichier house_1_df_ts : Foyer 1
file_info_h1 <- drive_find(pattern = "house_1_df_ts.csv")
file_id_h1 <- file_info_h1$id[1]
```

```r
house_1_df_dnb <- read.csv(drive_download(file_id_h1, type = "csv",
overwrite = TRUE)$local_path)

# Charger le fichier house_1_df_ts : Foyer 2
file_info_h2 <- drive_find(pattern = "house_2_df_ts.csv")
file_id_h2 <- file_info_h2$id[1]
house_2_df_dnb <- read.csv(drive_download(file_id_h2, type = "csv",
overwrite = TRUE)$local_path)

# Charger le fichier house_1_df_ts : Foyer 5
file_info_h5 <- drive_find(pattern = "house_5_df_ts.csv")
file_id_h5 <- file_info_h5$id[1]
house_5_df_dnb <- read.csv(drive_download(file_id_h5, type = "csv",
overwrite = TRUE)$local_path)

#### 5.2 - Afficher les bases de données
head(house_1_df_dnb, 5)
head(house_2_df_dnb, 5)
head(house_5_df_dnb, 5)

### **VI - TRAITEMENT ET ANALYSE EXPLORATION DES DONNEES**
sapply(house_1_df_dnb, class)
sapply(house_2_df_dnb, class)
sapply(house_5_df_dnb, class)

#### 6.2 - Convertir la variable unix en variable date
# Convertir la variable unix en variable de type date : Foyer 1
house_1_df_dnb$unix <- as.POSIXct(house_1_df_dnb$unix)
head(house_1_df_dnb, 5)

# Convertir la variable unix en variable de type date : Foyer 2
house_2_df_dnb$unix <- as.POSIXct(house_2_df_dnb$unix)
head(house_2_df_dnb, 5)

# Convertir la variable unix en variable de type date : Foyer 5
house_5_df_dnb$unix <- as.POSIXct(house_5_df_dnb$unix)
head(house_5_df_dnb, 5)

#### 6.3 - Création des variables Années, Mois, jour
# Création des variables explicatives temporelles : Foyer 1
house_1_df_dnb <- house_1_df_dnb %>%
  mutate(
    year = year(unix),
    month = month(unix),
    day = day(unix),
    hour = hour(unix)
  )
  head(house_1_df_dnb, 5)
```

```r
# Création des variables explicatives temporelles : Foyer 2
house_2_df_dnb <- house_2_df_dnb %>%
  mutate(
    year = year(unix),
    month = month(unix),
    day = day(unix),
    hour = hour(unix)
  )
  head(house_2_df_dnb, 5)

# Création des variables explicatives temporelles : Foyer 5
house_5_df_dnb <- house_5_df_dnb %>%
  mutate(
    year = year(unix),
    month = month(unix),
    day = day(unix),
    hour = hour(unix)
  )
  head(house_5_df_dnb, 5)

#### 6.3 - Encodage des variables
house_1_df_dnb <- house_1_df_dnb %>%
  arrange(unix) %>%
  mutate(
    active_power_lag1 = lag(active_power, 1),
    active_power_lag2 = lag(active_power, 2),
    hour_sin = sin(2 * pi * hour / 24),
    hour_cos = cos(2 * pi * hour / 24),
    month_sin = sin(2 * pi * month / 12 ),
    month_cos = cos(2 * pi * month / 12 )
  ) %>%
  na.omit()
head(house_1_df_dnb, 5)

house_2_df_dnb <- house_2_df_dnb %>%
  arrange(unix) %>%
  mutate(
    active_power_lag1 = lag(active_power, 1),
    active_power_lag2 = lag(active_power, 2),
    hour_sin = sin(2 * pi * hour / 24),
    hour_cos = cos(2 * pi * hour / 24),
    month_sin = sin(2 * pi * month / 12 ),
    month_cos = cos(2 * pi * month / 12 )
  ) %>%
  na.omit()
head(house_2_df_dnb, 5)
```

```
house_5_df_dnb <- house_5_df_dnb %>%
  arrange(unix) %>%
  mutate(
    active_power_lag1 = lag(active_power, 1),
    active_power_lag2 = lag(active_power, 2),
    hour_sin = sin(2 * pi * hour / 24),
    hour_cos = cos(2 * pi * hour / 24),
    month_sin = sin(2 * pi * month / 12 ),
    month_cos = cos(2 * pi * month / 12 )
  ) %>%
  na.omit()
head(house_5_df_dnb, 5)

# Sauvegarde des fichiers
write.csv2(house_1_df_dnb, file = "/content/RDa-
taset/house_1_df_dnb.csv", row.names = FALSE )
write.csv2(house_2_df_dnb, file = "/content/RDa-
taset/house_2_df_dnb.csv", row.names = FALSE )
write.csv2(house_5_df_dnb, file = "/content/RDa-
taset/house_5_df_dnb.csv", row.names = FALSE )

#### 6.4 - Normalisation des données
# Normalisation : Foyer 1
house_1_df_dbn_scaled <- house_1_df_dnb %>%
  mutate(across(c(active_power, apparent_power, main_rms_voltage),
scale))
write.csv2(house_1_df_dbn_scaled, file = "/content/RDa-
taset/house_1_df_dbn_scaled.csv", row.names = FALSE )

# Normalisation :
house_2_df_dbn_scaled <- house_2_df_dnb %>%
  mutate(across(c(active_power, apparent_power, main_rms_voltage),
scale))
write.csv2(house_2_df_dbn_scaled, file = "/content/RDa-
taset/house_2_df_dbn_scaled.csv", row.names = FALSE )

# Normalisation : Foyer 5
house_5_df_dbn_scaled <- house_5_df_dnb %>%
  mutate(across(c(active_power, apparent_power, main_rms_voltage),
scale))
write.csv2(house_5_df_dbn_scaled, file = "/content/RDa-
taset/house_5_df_dbn_scaled.csv", row.names = FALSE )

### **VII - TRAIN TEST SPLIT**
#### 7.1 - Foyer 1

# Séparer les données en 80% des données d'entrainement et 20% des don-
nées de test
```

```
house_1_train_size <- floor(0.80 * nrow(house_1_df_dbn_scaled))

# Base de données d'entrainement : Foyer 1
house_1_train_data <- house_1_df_dbn_scaled[1:house_1_train_size, ]
write.csv2(house_1_train_data, file = "/content/RDa-
taset/house_1_train_data.csv", row.names = FALSE )

# Base de données de test : Foyer 1
house_1_test_data <- house_1_df_dbn_scaled[(house_1_train_size +
1):nrow(house_1_df_dbn_scaled), ]
write.csv2(house_1_test_data, file = "/content/RDa-
taset/house_1_test_data.csv", row.names = FALSE )

#### 7.2 - Foyer 2

# Séparer les données en 80% des données d'entrainement et 20% des don-
nées de test
house_2_train_size <- floor(0.80 * nrow(house_2_df_dbn_scaled))

# Base de données d'entrainement : Foyer 2
house_2_train_data <- house_2_df_dbn_scaled[1:house_2_train_size, ]
write.csv2(house_2_train_data, file = "/content/RDa-
taset/house_2_train_data.csv", row.names = FALSE )

# Base de données de test : Foyer 2
house_2_test_data <- house_2_df_dbn_scaled[(house_2_train_size +
1):nrow(house_2_df_dbn_scaled), ]
write.csv2(house_2_test_data, file = "/content/RDa-
taset/house_2_test_data.csv", row.names = FALSE )

#### 7.3 - Foyer 3

# Séparer les données en 80% des données d'entrainement et 20% des don-
nées de test
house_5_train_size <- floor(0.80 * nrow(house_5_df_dbn_scaled))

# Base de données d'entrainement : Foyer 5
house_5_train_data <- house_5_df_dbn_scaled[1:house_5_train_size, ]
#write.csv2(house_5_train_data, file = "/content/RDa-
taset/house_5_train_data.csv", row.names = FALSE )

# Base de données de test : Foyer 5
house_5_test_data <- house_5_df_dbn_scaled[(house_5_train_size +
1):nrow(house_5_df_dbn_scaled), ]
write.csv2(house_5_test_data, file = "/content/RDa-
taset/house_5_test_data.csv", row.names = FALSE )

### **VIII - RESEAUX BAYESIENS DYNAMIQUES**
```

```
#### 8.1 - Foyer 1
house_1_dbn_learning <- house_1_train_data %>%
  select(active_power, apparent_power, main_rms_voltage, ac-
tive_power_lag1, active_power_lag2, hour_sin, hour_cos, month_sin,
month_cos)
head(house_1_dbn_learning, 5)
write.csv2(house_1_dbn_learning, file = "/content/RDa-
taset/house_1_dbn_learning.csv", row.names = FALSE )

# Apprentissage de la structure avec Hill-Climbing
structure_1 <- hc(house_1_dbn_learning, score = "bic-g")
plot(structure_1)  # Visualiser le graphe
ggsave("/content/RDataset/house_1_DynamicBayesianNetwork.png")

## Paramétrisation du Modèle
# Ajustement du DBN (modèle gaussien)
dbn_model_1 <- bn.fit(structure_1, data = house_1_dbn_learning, method
= "mle-g")
dbn_model_1

# Résumé des paramètres
print(dbn_model_1$active_power)


#### 8.2 - Foyer 2
house_2_dbn_learning <- house_2_train_data %>%
  select(active_power, apparent_power, main_rms_voltage, ac-
tive_power_lag1, active_power_lag2, hour_sin, hour_cos, month_sin,
month_cos)
head(house_2_dbn_learning, 5)
write.csv2(house_2_dbn_learning, file = "/content/RDa-
taset/house_2_dbn_learning.csv", row.names = FALSE )

# Apprentissage de la structure avec Hill-Climbing
structure_2 <- hc(house_2_dbn_learning, score = "bic-g")
plot(structure_2)  # Visualiser le graphe
ggsave("/content/RDataset/house_2_DynamicBayesianNetwork.png")

## Paramétrisation du Modèle
# Ajustement du DBN (modèle gaussien)
dbn_model_2 <- bn.fit(structure_2, data = house_2_dbn_learning, method
= "mle-g")
dbn_model_2

# Résumé des paramètres
print(dbn_model_2$active_power)
```

```
#### 8.3 - Foyer 3
house_5_dbn_learning <- house_5_train_data %>%
  select(active_power, apparent_power, main_rms_voltage, ac-
tive_power_lag1, active_power_lag2, hour_sin, hour_cos, month_sin,
month_cos)
head(house_5_dbn_learning, 5)
write.csv2(house_5_dbn_learning, file = "/content/RDa-
taset/house_5_dbn_learning.csv", row.names = FALSE )

# Apprentissage de la structure avec Hill-Climbing
structure_5 <- hc(house_5_dbn_learning, score = "bic-g")
plot(structure_5)  # Visualiser le graphe
ggsave("/content/RDataset/house_5_DynamicBayesianNetwork.png")

## Paramétrisation du Modèle
# Ajustement du DBN (modèle gaussien)
dbn_model_5 <- bn.fit(structure_5, data = house_5_dbn_learning, method
= "mle-g")
dbn_model_5

### **XIX - PREDICTION ET EVALUATION**
#### 9.1 - Foyer 1 --------------------------------
# Préparation des données de test
house_1_dbn_testing <- house_1_test_data %>%
  select(active_power, apparent_power, main_rms_voltage, ac-
tive_power_lag1, active_power_lag2, hour_sin, hour_cos, month_sin,
month_cos)
head(house_1_dbn_testing,5)
write.csv2(house_1_dbn_testing, file = "/content/RDa-
taset/house_1_dbn_testing.csv", row.names = FALSE )

# Prédiction
house_1_dbn_predictions <- predict(dbn_model_1, node = "active_power",
data = house_1_dbn_testing)
write.csv2(house_1_dbn_predictions, file = "/content/RDa-
taset/house_1_dbn_predictions.csv", row.names = FALSE )
head(house_1_dbn_predictions, 5)

# Calcul des métriques : Foyer 1
house_1_dbn_rmse <- sqrt(mean((house_1_dbn_testing$active_power -
house_1_dbn_predictions)^2))
house_1_dbn_mae <- mean(abs(house_1_dbn_testing$active_power -
house_1_dbn_predictions))
house_1_dbn_mape <- mean(abs((house_1_dbn_testing$active_power -
house_1_dbn_predictions) / house_1_dbn_testing$active_power)) * 100 #
S'assurer que les données ne contiennent pas des valeurs nulles ou man-
quantes
cat(
```

```
  "RMSE:",  house_1_dbn_rmse,
  "MAE:", house_1_dbn_mae,
  "MAPE:", house_1_dbn_mape
)

# Affichage des résultats
house_1_dbn_predictions_dataframe <- data.frame(
  time = house_1_test_data$unix,
  actual = house_1_dbn_testing$active_power,
  predicted = house_1_dbn_predictions
)
head(house_1_dbn_predictions_dataframe, 5)
write.csv2(house_1_dbn_predictions_dataframe, file="/content/RDa-
taset/house_1_dbn_predictions_dataframe.csv", row.names = FALSE)

# Visualisation
ggplot(house_1_dbn_predictions_dataframe, aes(x = time)) +
  geom_line(aes(y = actual, color = "Réel")) +
  geom_line(aes(y = predicted, color = "Prédit")) +
  labs(title = "Prédiction vs. Réel", x = "Temps", y = "Puissance Ac-
tive (W)") +
  scale_color_manual(values = c("Réel" = "blue", "Prédit" = "red"))
ggsave("/content/RDataset/house_1_bdn_actualvspredict.png")

# Log-likelihood
logLik_value <- logLik(dbn_model_1, data = house_1_dbn_learning)

# Nombre de paramètres (utile pour AIC/BIC)
num_params <- nparams(dbn_model_1)

# Nombre d'observations
n_obs <- nrow(house_1_dbn_learning)

# AIC et BIC
aic_value <- -2 * logLik_value + 2 * num_params
bic_value <- -2 * logLik_value + log(n_obs) * num_params

cat("Log-Likelihood:", logLik_value, "/n")
cat("AIC:", aic_value, "/n")
cat("BIC:", bic_value, "/n")

#### 9.1 - Foyer 2


# Préparation des données de test
house_2_dbn_testing <- house_2_test_data %>%
```

```
  select(active_power, apparent_power, main_rms_voltage, ac-
tive_power_lag1, active_power_lag2, hour_sin, hour_cos, month_sin,
month_cos)
head(house_2_dbn_testing,5)
write.csv2(house_2_dbn_testing, file = "/content/RDa-
taset/house_2_dbn_testing.csv", row.names = FALSE )

# Prédiction
house_2_dbn_predictions <- predict(dbn_model_2, node = "active_power",
data = house_2_dbn_testing)
write.csv2(house_2_dbn_predictions, file = "/content/RDa-
taset/house_2_dbn_predictions.csv", row.names = FALSE )
head(house_2_dbn_predictions, 5)

# Calcul des métriques : Foyer 1
house_2_dbn_rmse <- sqrt(mean((house_2_dbn_testing$active_power -
house_2_dbn_predictions)^2))
house_2_dbn_mae <- mean(abs(house_2_dbn_testing$active_power -
house_2_dbn_predictions))
house_2_dbn_mape <- mean(abs((house_2_dbn_testing$active_power -
house_2_dbn_predictions) / house_2_dbn_testing$active_power)) * 100 #
S'assurer que les données ne contiennent pas des valeurs nulles ou man-
quantes
cat(
  "RMSE:",  house_2_dbn_rmse,
  "MAE:", house_2_dbn_mae,
  "MAPE:", house_2_dbn_mape
)

# Visualisation des résultats
house_2_dbn_predictions_dataframe <- data.frame(
  time = house_2_test_data$unix,
  actual = house_2_dbn_testing$active_power,
  predicted = house_2_dbn_predictions
)
head(house_2_dbn_predictions_dataframe, 5)
write.csv2(house_2_dbn_predictions_dataframe, file="/content/RDa-
taset/house_2_dbn_predictions_dataframe.csv", row.names = FALSE)

# Visualisation
ggplot(house_2_dbn_predictions_dataframe, aes(x = time)) +
  geom_line(aes(y = actual, color = "Réel")) +
  geom_line(aes(y = predicted, color = "Prédit")) +
  labs(title = "Prédiction vs. Réel", x = "Temps", y = "Puissance Ac-
tive (W)") +
  scale_color_manual(values = c("Réel" = "blue", "Prédit" = "red"))
ggsave("/content/RDataset/house_2_bdn_actualvspredict.png")
```

```r
# Log-likelihood
logLik_value <- logLik(dbn_model_2, data = house_2_dbn_learning)

# Nombre de paramètres (utile pour AIC/BIC)
num_params <- nparams(dbn_model_2)

# Nombre d'observations
n_obs <- nrow(house_2_dbn_learning)

# AIC et BIC
aic_value <- -2 * logLik_value + 2 * num_params
bic_value <- -2 * logLik_value + log(n_obs) * num_params

cat("Log-Likelihood:", logLik_value, "/n")
cat("AIC:", aic_value, "/n")
cat("BIC:", bic_value, "/n")

#### 9.3 - Foyer 5
# Préparation des données de test
house_5_dbn_testing <- house_5_test_data %>%
  select(active_power, apparent_power, main_rms_voltage, ac-
tive_power_lag1, active_power_lag2, hour_sin, hour_cos, month_sin,
month_cos)
head(house_5_dbn_testing,5)
write.csv2(house_5_dbn_testing, file = "/content/RDa-
taset/house_5_dbn_testing.csv", row.names = FALSE )

# Prédiction
house_5_dbn_predictions <- predict(dbn_model_5, node = "active_power",
data = house_5_dbn_testing)
write.csv2(house_5_dbn_predictions, file = "/content/RDa-
taset/house_5_dbn_predictions.csv", row.names = FALSE )
head(house_5_dbn_predictions, 5)

# Calcul des métriques : Foyer 1
house_5_dbn_rmse <- sqrt(mean((house_5_dbn_testing$active_power -
house_5_dbn_predictions)^2))
house_5_dbn_mae <- mean(abs(house_5_dbn_testing$active_power -
house_5_dbn_predictions))
house_5_dbn_mape <- mean(abs((house_5_dbn_testing$active_power -
house_5_dbn_predictions) / house_5_dbn_testing$active_power)) * 100 #
S'assurer que les données ne contiennent pas des valeurs nulles ou man-
quantes
cat(
  "RMSE:",  house_5_dbn_rmse,
  "MAE:", house_5_dbn_mae,
  "MAPE:", house_5_dbn_mape
)
```

```r
# Visualisation des résultats
house_5_dbn_predictions_dataframe <- data.frame(
  time = house_5_test_data$unix,
  actual = house_5_dbn_testing$active_power,
  predicted = house_5_dbn_predictions
)
head(house_5_dbn_predictions_dataframe, 5)
write.csv2(house_5_dbn_predictions_dataframe, file="/content/RDataset/house_5_dbn_predictions_dataframe.csv", row.names = FALSE)

# Visualisation
ggplot(house_5_dbn_predictions_dataframe, aes(x = time)) +
  geom_line(aes(y = actual, color = "Réel")) +
  geom_line(aes(y = predicted, color = "Prédit")) +
  labs(title = "Prédiction vs. Réel", x = "Temps", y = "Puissance Active (W)") +
  scale_color_manual(values = c("Réel" = "blue", "Prédit" = "red"))
ggsave("/content/RDataset/house_5_bdn_actualvspredict.png")
# Log-likelihood
logLik_value <- logLik(dbn_model_5, data = house_5_dbn_learning)

# Nombre de paramètres (utile pour AIC/BIC)
num_params <- nparams(dbn_model_5)

# Nombre d'observations
n_obs <- nrow(house_5_dbn_learning)

# AIC et BIC
aic_value <- -2 * logLik_value + 2 * num_params
bic_value <- -2 * logLik_value + log(n_obs) * num_params

cat("Log-Likelihood:", logLik_value, "/n")
cat("AIC:", aic_value, "/n")
cat("BIC:", bic_value, "/n")
```