



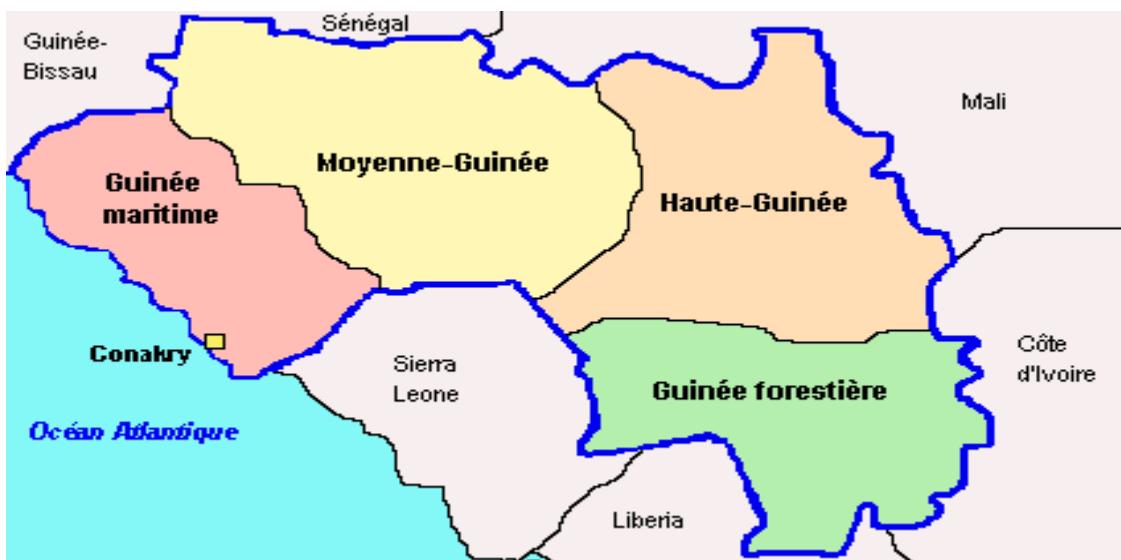
<b>Étudiant</b>	Mamadou Billo Diallo
<b>Cours</b>	8IN309, 8IN319, 8IN329, 8IN339
<b>Remise Jalon 5</b>	Final
<b>Date</b>	2024-12-14
<b>Professeur</b>	Haïfa Nakouri

Prédiction de l'Irrigation (Besoin en eau) en agriculture .....	4
Business understanding.....	4
Contexte et Enjeux.....	4
Objectifs du Projet.....	4
Défis.....	4
Opportunités .....	5
Processus pour Calculer le Besoin en Eau des Cultures afin de trouver notre variable cible. ....	5
1-Collecte des Données Climatiques et Agronomiques .....	5
2- Calcul de l'Évapotranspiration de Référence ( $ET_0$ ) .....	6
Un peu d'histoire : .....	7
3- Application des Coefficients de Culture ( $K_c$ ) .....	7
4- Prise en Compte des Précipitations :.....	9
5- Variable cible : .....	9
Conclusion .....	9
Compréhension des données.....	9
Caractéristique des variables :.....	11
Niveau.....	11
Exploration des données pour plus de clarté : .....	12
Types des variables :.....	12
Statistiques et Interprétation : .....	13
Visualisation :.....	15
Corrélation entre variable : .....	16
Compréhension de la tendance des données par jour, mois année et par saison : .....	18
Par jour : .....	18
Par mois : .....	19
Par Année : .....	20
Par saison : .....	21
Préparation des données.....	22
Importation des données et organisation : .....	22
Traitement appliqué pour la visualisation expliqué dans la partie compréhension des données : .....	23
Par jour : .....	23
Par mois : .....	23

Par an :.....	23
Par saison :.....	23
Doublons.....	24
Calcul de la variable Target IRR :.....	24
Corrélation des variables par rapport à IRR : .....	26
Normalisation des données :.....	26
Split et gestions des valeurs manquantes ainsi que des outliers : .....	27
Entrainement Modèles :.....	31
Evaluation des modèles :.....	45
Conclusion .....	51
Déploiement.....	52
References .....	53

# Prédiction de l'Irrigation (Besoin en eau) en agriculture

## Business understanding



### Contexte et Enjeux

La région de la Moyenne Guinée, également connue sous le nom de Fouta Djallon, est caractérisée par des hauts plateaux et des conditions climatiques semi-arides, avec une forte variabilité des précipitations entre les saisons sèches et humides. L'agriculture dans cette région repose principalement sur la culture de pommes de terre, de maïs, de manioc, de riz, de banane et de tant d'autre avec une dépendance importante aux précipitations naturelles. Cependant, le changement climatique et l'irrégularité croissante des saisons pluviométriques posent des défis majeurs en termes de gestion des ressources en eau et d'optimisation de l'irrigation.

### Objectifs du Projet

Le projet vise à développer un système intelligent de gestion de l'irrigation pour maximiser la production agricole pour la culture de la banane dans la région de la Moyenne Guinée. Grâce à l'utilisation de données climatiques locales (température, humidité, précipitations, vitesse du vent) et de données de sol, l'objectif est de prédire avec précision les besoins en eau de cette culture afin d'optimiser l'irrigation tout en réduisant le gaspillage des ressources en eau.

### Défis

- Variabilité climatique : Les précipitations dans la Moyenne Guinée sont irrégulières et imprévisibles, ce qui complique la planification de l'irrigation.

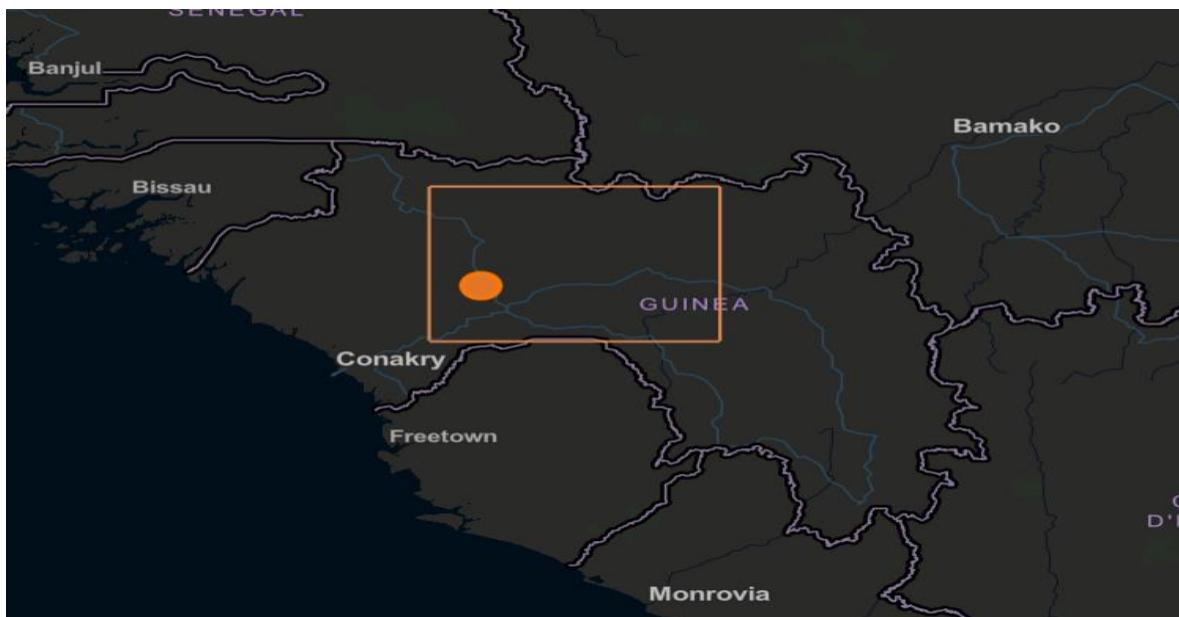
- Limitation des infrastructures d'irrigation : Les systèmes d'irrigation actuels sont souvent rudimentaires et manquent de technologie intelligente pour s'adapter aux besoins dynamiques des cultures.
- Disponibilité des données : Les données en temps réel sur les conditions climatiques et l'humidité du sol sont peu disponibles ou incomplètes dans certaines parties de la région.

## Opportunités

- Amélioration du rendement agricole : Un système d'irrigation optimisé peut réduire le stress hydrique sur les cultures pendant les périodes sèches, ce qui améliorera les rendements des cultures locales.
- Efficacité dans l'utilisation de l'eau : En ajustant l'irrigation en fonction des prévisions météorologiques et des besoins en eau des plantes, il est possible de réduire le gaspillage d'eau tout en maintenant une production agricole élevée.

**Processus pour Calculer le Besoin en Eau des Cultures afin de trouver notre variable cible.**

### 1-Collecte des Données Climatiques et Agronomiques



Nous commencerons par recueillir les données nécessaires sur les paramètres météorologiques et les caractéristiques des cultures. Les données collectées incluront :

- Données climatiques en temps réel : Ces données proviendront des sources comme la NASA, couvrant la température, l'humidité relative, la vitesse du vent, et le rayonnement solaire.
- Données sur le sol : Nous collecterons des informations sur l'humidité du sol et ses caractéristiques physiques.

- Données sur les cultures : Il s'agit des coefficients de culture ( $K_c$ ), des caractéristiques spécifiques de chaque culture (pommes de terre, maïs, manioc, riz, etc.), et des phases de croissance des plantes obtenus par le site de FAO.

## 2- Calcul de l'Évapotranspiration de Référence ( $ET_0$ )

Le besoin en eau des cultures est basé sur l'évapotranspiration de référence ( $ET_0$ ). Cette mesure prend en compte les conditions météorologiques et donne une estimation de la quantité d'eau évaporée et transpirée par une surface de référence (gazon ou culture de référence). La méthode de Penman-Monteith est utilisée pour calculer l' $ET_0$  à partir des paramètres suivants :

- LAT (Latitude)
- LON (Longitude)
- YEAR (Année)
- DOY (Jour de l'année - Day of Year)
- T2M (Température à 2 mètres, en °C)
- T2MDEW (Point de rosée à 2 mètres, en °C)
- T2M\_MAX (Température maximale à 2 mètres, en °C)
- T2M\_MIN (Température minimale à 2 mètres, en °C)
- ALLSKY\_SFC\_SW\_DWN (Rayonnement solaire descendant à la surface, en MJ/m<sup>2</sup>/jour)
- ALLSKY\_SFC\_LW\_DWN (Rayonnement infrarouge descendant à la surface, en W/m<sup>2</sup>)
- PS (Pression atmosphérique à la surface, en kPa)
- WS2M (Vitesse du vent à 2 mètres, en m/s)

La formule de Penman-Monteith est la suivante :

$$ET_0 = \frac{0.408\Delta(R_n - G) + \gamma \frac{900}{T+273} u_2(e_s - e_a)}{\Delta + \gamma(1 + 0.34u_2)}$$

	mm/day
--	--------

Où :

- $\Delta$  = pente de la courbe de pression de vapeur saturante.
- $R_n$  = rayonnement net à la surface (W/m<sup>2</sup>).
- $G$  = flux de chaleur dans le sol (en général, négligé).
- $\gamma$  = constante psychrométrique.
- $T$  = température de l'air à 2 mètres.
- $u_2$  = vitesse du vent à 2 mètres.
- $e_s - e_a$  = pression de vapeur saturante et réelle.

Pour obtenir des informations détaillées sur comment chaque paramètre est obtenu veuillez visiter ce lien : Chapter 3 - Meteorological data (fao.org)

Vous trouverez sur ce lien comment est calculé ET<sub>0</sub> : Chapter 4 - Determination of ET<sub>0</sub> (fao.org)

#### Un peu d'histoire :

L'équation de Penman-Monteith est une formule utilisée pour estimer l'évapotranspiration de référence, c'est-à-dire la quantité d'eau perdue par évaporation et transpiration d'une surface végétale.

- En 1948, Penman a combiné deux méthodes : le bilan énergétique (quantité d'énergie disponible pour l'évaporation) et la méthode de transfert de masse (transport d'humidité par le vent). Il a ainsi dérivé une équation permettant de calculer l'évaporation d'une surface d'eau libre à partir de données climatiques comme la température, l'humidité, la vitesse du vent et l'ensoleillement.
- Par la suite, cette méthode a été améliorée et adaptée aux surfaces végétales en introduisant des concepts de résistance. Ces résistances sont de deux types :
  - La résistance aérodynamique ( $r_a$ ), qui décrit la friction de l'air autour des plantes.
  - La résistance de surface ( $r_s$ ), qui concerne la résistance à l'évaporation des feuilles et du sol.
- En 1990, la FAO (Organisation des Nations Unies pour l'Alimentation et l'Agriculture) a organisé une réunion d'experts pour revoir les méthodes de calcul de l'évapotranspiration des cultures. Ils ont recommandé la méthode de Penman-Monteith comme standard pour calculer l'évapotranspiration de référence. Ils ont défini une culture de référence (herbe verte, uniformément arrosée) pour rendre les calculs universels et fiables.

Ainsi, la méthode de Penman-Monteith est devenue la norme mondiale pour estimer les besoins en eau des cultures, fournissant des résultats plus cohérents que les méthodes précédentes.

#### 3- Application des Coefficients de Culture (K<sub>c</sub>)

Une fois que l'évapotranspiration de référence (ET<sub>0</sub>) est calculée, elle doit être ajustée pour chaque type de culture en fonction de son stade de croissance. Chaque culture a un coefficient de culture (K<sub>c</sub>), qui varie en fonction de la phase de croissance (initiale, intermédiaire, et finale).

Le besoin en eau spécifique des cultures (ET<sub>c</sub>) est calculé comme suit : ET<sub>c</sub>=K<sub>c</sub> \* ET<sub>0</sub>

Où :

- K<sub>c</sub> est le coefficient de culture, déterminé pour chaque culture et phase de croissance.
- ET<sub>0</sub> est l'évapotranspiration de référence.

Voici quelques coefficient culturaux (K<sub>c</sub>) fourni par la FAO :

**ANNEXE 1**  
**Coefficients cultureaux pour divers stades de croissance**

	Stade de Croissance			
	Initial	Développement	Mi-saison	Tardif
Agrume	0.75	0.75	0.75	0.75
Arachide	0.40	0.78	1.15	0.60
Banane	1.00	1.00	1.00	1.00
Betterave sucrière	0.35	0.78	1.20	0.70
Blé	0.40	0.78	1.15	0.30
Cacao	1.00	1.00	1.00	1.00
Café	1.00	1.00	1.00	1.00
Canne à sucre	1.00	1.00	1.00	1.00
Caoutchouc	1.00	1.00	1.00	1.00
Coton	0.35	0.78	1.20	0.60
Colza	0.35	0.73	1.10	0.35
Cultures oléagineuses	0.35	0.75	1.15	0.35
Fibres textiles	0.35	0.68	1.00	0.60
Fourrage	0.40	0.70	1.00	0.90
Fruit	0.75	0.75	0.75	0.75
Légumes	0.60	0.85	1.10	0.90
Légumes secs	0.40	0.78	1.15	0.55
Mais	0.30	0.75	1.20	0.60
Manioc	0.60	0.85	1.10	0.90
Millet	0.30	0.65	1.00	0.30
Noix de coco	1.00	1.00	1.00	1.00
Orge	0.30	0.73	1.15	0.25
Palme	1.00	1.00	1.00	1.00
Patate douce	1.00	1.00	1.00	1.00
Plantain	1.00	1.00	1.00	1.00
Pomme de terre	0.50	0.83	1.15	0.75
Riz	1.20	1.15	1.10	0.80
Sésame	0.35	0.73	1.10	0.25
Soja	0.40	0.78	1.15	0.50
Sorgho	0.30	0.70	1.10	0.55
Tabac	0.50	0.85	1.20	0.80
Thé	1.05	1.05	1.05	1.05
Autres racines	1.00	1.00	1.00	1.00
Autres céréales	1.00	1.00	1.00	1.00
Tournesol	0.35	0.73	1.10	0.35

Pourquoi le coefficient cultural de la banane est-il de 1 ?

Le coefficient cultural de la banane, fixé à 1, reflète la réalité que cette plante exige une quantité d'eau importante et stable tout au long de son cycle de croissance. Son évapotranspiration est similaire à celle d'une herbe de référence, ce qui traduit ses besoins élevés en eau. Cette constance s'explique par sa croissance continue et la structure même de la banane, typique des cultures tropicales, qui nécessite une irrigation régulière pour maintenir un développement optimal.

Comment aurait-on procédé pour une autre culture, comme le coton, avec des coefficients variables : initial (0,35), développement (0,78), mi-saison (1,20), tardif (0,60) ?

C'est une excellente question qui introduit une complexité supplémentaire dans l'analyse des données. Pour des cultures comme le coton, où les besoins en eau varient selon les phases de croissance, nous devons adapter nos calculs en conséquence. Par exemple, nous pourrions créer une variable  $kc$  et simuler la culture en prenant en compte les phases de développement. Si nous imaginons le début de la culture en mai ou juin, mois propices en Guinée, nous devons déterminer combien de jours durent les phases initiales, de développement, de mi-saison et tardives. À partir de ces informations, nous calculerions l'irrigation nécessaire pour chaque phase. Le processus se répéterait jusqu'en 2024, en réajustant à chaque phase tardive, puis en recommençant au stade initial l'année suivante. L'important ici est que ce modèle s'applique à une zone spécifique, et non à toutes les régions. Ainsi, pour un agriculteur qui commence sa culture en mai, il suivra ce cycle, avec des ajustements basés sur les phases de croissance de son

coton. Ce processus continu nous permet d'adapter dynamiquement le coefficient cultural et de prévoir les besoins en eau pour chaque cycle.

#### 4- Prise en Compte des Précipitations :

Le besoin en irrigation sera ajusté en tenant compte des précipitations naturelles et de la capacité du sol à retenir l'eau. Si les précipitations enregistrées couvrent partiellement ou totalement les besoins en eau des cultures, l'irrigation peut être réduite.

La quantité d'irrigation nécessaire (Irr) est calculée comme suit :  $Irr = ETc - P$

Où :

- $P$  est la quantité d'eau apportée par les précipitations (en mm).
- Si  $P > ETc$ , l'irrigation peut être nulle, ou l'excès est utilisé pour la recharge en eau du sol.

une fois que nous avons déterminé l'évapotranspiration de la plante ( $ETc$ ), il est essentiel de savoir combien de millilitres de pluie ( $P$ ) sont tombés ce jour-là. En soustrayant la précipitation de l' $ETc$ , nous obtenons la quantité d'eau que la pluie peut diminuer dans notre irrigation . Si la pluie couvre une partie des besoins en eau, il n'est pas nécessaire d'irriguer toute l'évapotranspiration. Cette méthode garantit une gestion précise de l'irrigation

#### Prise en Compte de l'humidité du sol :

Il est essentiel de prendre en compte l'humidité du sol. Si une plante a besoin d'eau et que le sol peut en fournir une partie, on n'irrigue que la différence nécessaire. Dans les données de la NASA, il existe une variable qui indique le pourcentage d'humidité du sol. En la multipliant par l'évapotranspiration, on obtient la quantité d'eau que le sol peut fournir par rapport à l'évapotranspiration totale, ce qui permet d'ajuster l'irrigation en conséquence.

#### 5- Variable cible :

Après avoir calculé le besoin en eau de notre culture pour les données climatiques qu'on a obtenu de la NASA on va stocker ces résultats dans une variable et celle-ci deviendra la variable cible que nous tenterons de prédire dans le cadre de notre étude pour les données en temps réelles.

## Conclusion

La mise en œuvre d'un système de gestion intelligente de l'irrigation pour la Moyenne Guinée permettrait d'améliorer la résilience des agriculteurs face aux fluctuations climatiques tout en optimisant l'utilisation de l'eau. L'analyse des données climatiques et du sol aidera à prédire avec précision les besoins en irrigation, contribuant ainsi à la sécurité alimentaire de la région.

## Compréhension des données

#### Information concernant notre base de données :

Notre base de données est structurée en time series parce qu'il stock hebdomadairement des données en temps réelles basée sur des données comme la température , l'humidité etc...

NASA/POWER CERES/MERRA2 Native Resolution Daily Data	
Dates (month/day/year): 01/01/2020 through 09/30/2024	
Location: Regional	
Elevation from MERRA-2: Average for 0.5 x 0.625 degree lat/lon region = na meters	
The value for missing source data that cannot be computed or is outside of the sources availability range: -999	
Parameter(s):	
T2M	MERRA-2 Temperature at 2 Meters (C)
T2MDEW	MERRA-2 Dew/Frost Point at 2 Meters (C)
T2M_MAX	MERRA-2 Temperature at 2 Meters Maximum (C)
T2M_MIN	MERRA-2 Temperature at 2 Meters Minimum (C)
ALLSKY_SFC_SW_DWN	CERES SYN1deg All Sky Surface Shortwave Downward Irradiance (MJ/m^2/day)
ALLSKY_SFC_LW_DWN	CERES SYN1deg All Sky Surface Longwave Downward Irradiance (W/m^2)
PS	MERRA-2 Surface Pressure (kPa)
WS2M	MERRA-2 Wind Speed at 2 Meters (m/s)
PRECTOTCORR	MERRA-2 Precipitation Corrected (mm/day)
GWETROOT	MERRA-2 Root Zone Soil Wetness (1)

Les données nous proviennent de la NASA/POWER CERES/MERRA2 NATIVE Resolution Daily Data.

Nous observons que toutes données qui est -999 est une donnée manquante.

### Variables :

Nous avons considéré ces différentes variables obtenues à partir de la NASA :

LAT	LON	YEAR	DOY	T2M	T2MDEW	T2M_MAX	T2M_MIN
ALLSKY_SFC_SW_DWN	ALLSKY_SFC_LW_DWN	PS	WS2M				

qui sont en liens avec la formule de Penman-Monteith.

Ainsi que PRECTOTCORR et GWETROOT pour la précipitation et l'humidité du sol respectivement.

### Une question très intéressante est de savoir pourquoi avoir considéré ces variables?

Sur le site de la FAO on retrouve tout le processus pour le calcul de l'évapotranspiration ainsi que les variables climatiques qu'il faut utiliser.

#### Calculation procedure

##### Calculation sheet

$ET_0$  can be estimated by means of the calculation sheet presented in Box 11. The calculation sheet refers to tables in Annex II for the determination of some of the climatic parameters. The calculation procedure consists of the following steps:

- Derivation of some climatic parameters from the daily maximum ( $T_{\max}$ ) and minimum ( $T_{\min}$ ) air temperature, altitude (z) and mean wind speed ( $u_2$ ).
- Calculation of the vapour pressure deficit ( $e_s - e_a$ ). The saturation vapour pressure ( $e_s$ ) is derived from  $T_{\max}$  and  $T_{\min}$ , while the actual vapour pressure ( $e_a$ ) can be derived from the dewpoint temperature ( $T_{dew}$ ), from maximum ( $RH_{\max}$ ) and minimum ( $RH_{\min}$ ) relative humidity, from the maximum ( $RH_{\max}$ ), or from mean relative humidity ( $RH_{mean}$ ).
- Determination of the net radiation ( $R_n$ ) as the difference between the net shortwave radiation ( $R_{ns}$ ) and the net longwave radiation ( $R_{nl}$ ). In the calculation sheet, the effect of soil heat flux (G) is ignored for daily calculations as the magnitude of the flux in this case is relatively small. The net radiation, expressed in  $MJ \cdot m^{-2} \cdot day^{-1}$ , is converted to mm/day (equivalent evaporation) in the FAO Penman-Monteith equation by using 0.408 as the conversion factor within the equation.
- $ET_0$  is obtained by combining the results of the previous steps.

## Caractéristique des variables :

- **LAT (Latitude)** : Coordonnée géographique nord-sud, influençant le climat d'une région.
- **LON (Longitude)** : Coordonnée est-ouest, utilisée pour localiser précisément une région.
- **YEAR (Année)** : Indique l'année de collecte des données.
- **DOY (Day of Year, Jour de l'Année)** : Numéro du jour dans l'année, important pour suivre les variations saisonnières.
- **T2M (Température à 2 mètres)** : Température de l'air à 2 mètres du sol, en degrés Celsius (°C). C'est une mesure importante pour l'évapotranspiration, influençant les besoins en eau.
- **T2MDEW (Point de rosée à 2 mètres)** : Température à laquelle l'air est saturé d'humidité, mesurée en °C. Cela aide à comprendre l'humidité ambiante.
- **T2M\_MAX (Température maximale à 2 mètres)** : Température maximale enregistrée à 2 mètres du sol en °C, mesurant les journées les plus chaudes.
- **T2M\_MIN (Température minimale à 2 mètres)** : Température minimale enregistrée à 2 mètres du sol en °C, influençant les conditions nocturnes.
- **ALLSKY\_SFC\_SW\_DWN (Rayonnement solaire direct à la surface)** : Irradiance solaire reçue à la surface, en mégajoules par mètre carré par jour (MJ/m<sup>2</sup>/jour), un indicateur de l'énergie solaire disponible pour l'évaporation.
- **ALLSKY\_SFC\_LW\_DWN (Rayonnement infrarouge à la surface)** : Irradiance infrarouge mesurée en watts par mètre carré (W/m<sup>2</sup>), influençant la température du sol et l'évaporation.
- **PS (Pression de surface)** : Pression atmosphérique mesurée en kilopascals (kPa), utile pour comprendre les conditions météorologiques locales.
- **WS2M (Vitesse du vent à 2 mètres)** : Vitesse du vent à 2 mètres du sol, en mètres par seconde (m/s). Des vents plus forts augmentent l'évaporation et les besoins en irrigation.
- **PRECTOTCORR (Précipitations corrigées)** : Précipitations corrigées mesurées en millimètres par jour (mm/jour). C'est un facteur clé pour estimer la quantité d'eau apportée naturellement aux cultures.
- **GWETROOT (Humidité du sol dans la zone racinaire)** : Taux de saturation du sol dans la zone des racines (de 0 à 1). Il indique la quantité d'eau disponible dans le sol pour les plantes, influençant directement les besoins en irrigation.

## Niveau

Il est important de comprendre le niveau d'urgence de l'irrigation ceci pourrait nous permettre de bien comprendre à quel moment il faut irriguer. voici quelque point indicateur de l'urgence dans la prise de décision relié à l'irrigation.

- **Aucun besoin d'irrigation** : si IRR présente des valeurs comme 0 on n'a pas besoin d'irriguer. S'il a des valeurs négative ça veut dire qu'avec les pluies et l'humidité du sol la plante arrive à bien être irriguer.
- **Faible besoin d'irrigation** : plus la valeur de l'irrigation est proche de 0 moins l'urgence pour irriguer sera forte.
- **Besoin élevé d'irrigation** : plus la valeur s'éloigne de 0 positivement plus l'urgence d'irrigation augmentera.

## Exploration des données pour plus de clarté :

#	LAT	LON	YEAR	DOY	T2M	T2MDEW	T2M_MAX	T2M_MIN	ALLSKY_SFC_SW_DWN	ALLSKY_SFC_LW_DWN	PS	WS2M	PRECTOTCORR	GWETROOT
0	10.25	-13.25	2020	1	24.87	11.90	33.43	17.74	20.67	360.43	98.03	1.61	0.00	0.53
1	10.25	-12.75	2020	1	24.53	8.15	34.17	16.60	21.14	327.40	97.20	1.98	0.00	0.54
2	10.25	-12.25	2020	1	23.35	6.59	33.13	15.52	21.14	327.40	96.19	2.45	0.00	0.54
3	10.25	-11.75	2020	1	21.30	7.34	30.44	13.88	21.47	315.98	95.10	2.67	0.00	0.53
4	10.25	-11.25	2020	1	19.97	8.64	28.90	11.97	21.47	315.98	95.09	2.31	0.00	0.56
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
55515	11.75	-11.75	2024	274	24.65	22.27	28.40	21.43	16.71	415.40	94.77	1.40	3.64	NaN
55516	11.75	-11.25	2024	274	25.08	22.39	28.75	21.83	16.71	415.40	95.20	1.41	3.07	NaN
55517	11.75	-10.75	2024	274	25.55	22.63	29.42	22.00	20.66	423.88	95.84	1.37	2.84	NaN
55518	11.75	-10.25	2024	274	25.66	22.76	29.48	21.93	20.66	423.88	96.12	1.38	2.91	NaN
55519	11.75	-9.75	2024	274	25.79	22.91	29.53	21.92	20.29	422.62	96.35	1.38	2.87	NaN

55520 rows x 14 columns

Voici un aperçu de notre base de données avec 55 520 lignes pour 14 paramètres qui sont nos variables.

Nous voyons toutes les variables dont on a discuté plus haut avec leurs valeurs journalières respectives de 2020 à 2024.

Au niveau de la précipitation(PRECTOTCORR) on observe des données égale à 0. Ces valeurs nous indique qu'il n'a pas plu cette journée.

## Types des variables :

# on vérifié nos variables				
df.info()				
<class 'pandas.core.frame.DataFrame'>				
RangeIndex: 55520 entries, 0 to 55519				
Data columns (total 14 columns):				
#	Column	Non-Null Count	Dtype	
0	LAT	55520	non-null	float64
1	LON	55520	non-null	float64
2	YEAR	55520	non-null	int64
3	DOY	55520	non-null	int64
4	T2M	55424	non-null	float64
5	T2MDEW	55424	non-null	float64
6	T2M_MAX	55424	non-null	float64
7	T2M_MIN	55424	non-null	float64
8	ALLSKY_SFC_SW_DWN	55520	non-null	float64
9	ALLSKY_SFC_LW_DWN	55520	non-null	float64
10	PS	55424	non-null	float64
11	WS2M	55424	non-null	float64
12	PRECTOTCORR	55424	non-null	float64
13	GWETROOT	52608	non-null	float64

dtypes: float64(12), int64(2)  
memory usage: 5.9 MB

On observe plus d'information sur le type des données qui sont pour la plupart des float64 sauf pour les variables année et jours qui sont des int64. On peut voir qu'avec 55 520 lignes au total il y'a certaines variables qui ont des valeurs manquantes. On essaiera de les gérer dans le prochain chapitre.

#### Statistiques et Interprétation :

		count	mean	std	min	25%	50%	75%	max
1 s	LAT	55520.0	11.000000	0.559022	10.25	10.6250	11.00	11.3750	11.75
	LON	55520.0	-11.500000	1.145654	-13.25	-12.3750	-11.50	-10.6250	-9.75
	YEAR	55520.0	2021.893948	1.372968	2020.00	2021.0000	2022.00	2023.0000	2024.00
	DOY	55520.0	175.919885	103.085622	1.00	87.0000	174.00	261.0000	366.00
	T2M	55424.0	25.960243	2.772404	17.53	23.9500	25.24	27.9300	36.24
	T2MDEW	55424.0	16.903887	6.648823	-8.34	12.7000	20.18	22.0400	25.17
	T2M_MAX	55424.0	31.947039	4.533371	21.70	28.2900	30.61	35.8825	45.20
	T2M_MIN	55424.0	20.856558	2.512726	9.81	19.7275	21.30	22.4700	28.85
	ALLSKY_SFC_SW_DWN	55520.0	20.106178	3.826628	1.25	17.8100	20.88	22.8300	28.27
	ALLSKY_SFC_LW_DWN	55520.0	387.784214	28.920877	266.20	372.6850	396.07	407.6000	451.89
	PS	55424.0	95.568458	1.323025	92.37	94.5300	95.73	96.3400	98.90
	WS2M	55424.0	1.779947	0.575334	0.45	1.3400	1.71	2.1500	4.72
	PRECTOTCORR	55424.0	7.454662	20.278273	0.00	0.0000	0.93	7.6525	848.57
	GWETROOT	52608.0	0.668956	0.204935	0.40	0.4800	0.60	0.8800	1.00

#### Pour DOY

- Symétrie : La moyenne (175.92) est inférieure à la médiane (174.00), suggérant une asymétrie positive.
- Dispersion : L'écart-type (103.09) est élevé, indiquant une grande variabilité dans les jours de l'année. L'IQR (174.00 à 261.00) montre une large dispersion.
- Outliers : Les valeurs minimale (1.00) et maximale (366.00) montrent un écart significatif, indiquant la présence de valeurs extrêmes.

#### Pour T2M

- Symétrie : La moyenne (25.96) est légèrement supérieure à la médiane (25.24), ce qui peut indiquer une légère asymétrie négative.
- Dispersion : L'écart-type (2.772) est modéré, montrant une dispersion raisonnable des températures. L'IQR (3.98) indique que les valeurs sont relativement concentrées autour de la médiane.
- Outliers : Les valeurs minimale (17.53) et maximale (36.24) ne s'éloignent pas trop des quartiles, indiquant peu ou pas de valeurs extrêmes.

#### Pour T2MDEW

- Symétrie : La moyenne (16.90) est inférieure à la médiane (20.18), suggérant une asymétrie positive.
- Dispersion : L'écart-type (6.649) est élevé, indiquant une variabilité significative des températures de rosée. L'IQR (9.34) montre une large dispersion des valeurs.

- Outliers : Les valeurs minimale (-8.34) et maximale (25.17) montrent un écart considérable par rapport aux quartiles, indiquant des valeurs extrêmes.

#### **Pour T2M\_MAX**

- Symétrie : La moyenne (31.95) est légèrement inférieure à la médiane (30.61), suggérant une légère asymétrie positive.
- Dispersion : L'écart-type (4.533) est modéré, indiquant une dispersion raisonnable des valeurs maximales de température. L'IQR (7.59) montre une dispersion modérée.
- Outliers : Les valeurs minimale (21.70) et maximale (45.20) ne s'éloignent pas beaucoup des quartiles, indiquant peu ou pas de valeurs extrêmes.

#### **Pour T2M\_MIN**

- Symétrie : La moyenne (20.86) est légèrement inférieure à la médiane (21.30), ce qui suggère une légère asymétrie négative.
- Dispersion : L'écart-type (2.513) est faible, indiquant que les valeurs minimales de température sont concentrées autour de la moyenne. L'IQR (2.74) est également faible.
- Outliers : Les valeurs minimale (9.81) et maximale (28.85) ne montrent pas de valeurs extrêmes significatives.

#### **Pour ALLSKY\_SFC\_SW\_DWN**

- Symétrie : La moyenne (20.11) est légèrement inférieure à la médiane (20.88), suggérant une légère asymétrie positive.
- Dispersion : L'écart-type (3.827) est modéré, indiquant une dispersion raisonnable des données. L'IQR (5.02) montre une dispersion modérée.
- Outliers : Les valeurs minimale (1.25) et maximale (28.27) montrent un écart significatif par rapport aux quartiles, indiquant des valeurs extrêmes.

#### **Pour ALLSKY\_SFC\_LW\_DWN**

- Symétrie : La moyenne (387.78) est légèrement inférieure à la médiane (396.07), suggérant une légère asymétrie positive.
- Dispersion : L'écart-type (28.921) est élevé, indiquant une grande variabilité des valeurs. L'IQR (34.915) montre également une large dispersion.
- Outliers : Les valeurs minimale (266.20) et maximale (451.89) montrent un écart significatif, indiquant des valeurs extrêmes.

#### **Pour PS**

- Symétrie : La moyenne (95.57) est très proche de la médiane (95.73), suggérant une distribution symétrique.
- Dispersion : L'écart-type (1.323) est faible, indiquant que les valeurs de pression sont concentrées autour de la moyenne. L'IQR (1.81) confirme cette faible dispersion.
- Outliers : Les valeurs minimale (92.37) et maximale (98.90) ne s'éloignent pas beaucoup des quartiles, suggérant peu ou pas de valeurs extrêmes.

#### **Pour WS2M**

- Symétrie : La moyenne (1.78) est légèrement supérieure à la médiane (1.71), ce qui peut indiquer une légère asymétrie négative.
- Dispersion : L'écart-type (0.575) est faible, indiquant que les vitesses de vent sont proches de la moyenne. L'IQR (0.81) suggère également une faible dispersion.
- Outliers : Les valeurs minimale (0.45) et maximale (4.72) montrent un écart plus large par rapport aux quartiles, indiquant la présence de valeurs extrêmes.

**Pour PRECTOTCORR**

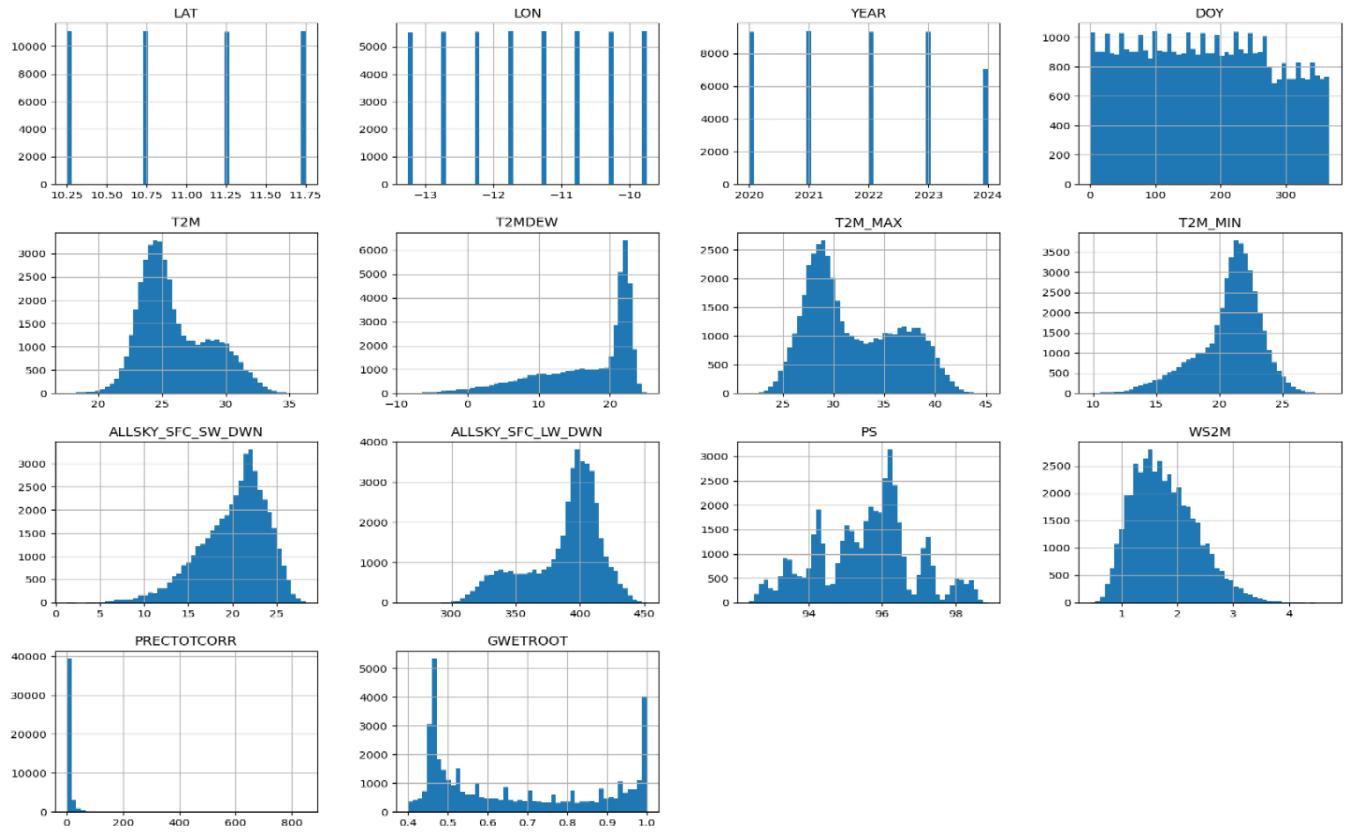
- Symétrie : La moyenne (7.45) est inférieure à la médiane (0.93), ce qui suggère une asymétrie positive significative.
- Dispersion : L'écart-type (20.278) est élevé, indiquant une très grande variabilité. L'IQR (6.6825) montre également une large dispersion.
- Outliers : Les valeurs minimale (0.00) et maximale (848.57) montrent un écart très significatif, suggérant de nombreux outliers.

**Pour GWETROOT**

- Symétrie : La moyenne (0.67) est légèrement supérieure à la médiane (0.60), indiquant une légère asymétrie négative.
- Dispersion : L'écart-type (0.205) est faible, montrant que les valeurs sont relativement concentrées autour de la moyenne. L'IQR (0.4) indique également une faible dispersion.
- Outliers : Les valeurs minimale (0.40) et maximale (1.00) ne montrent pas de valeurs extrêmes significatives.

Visualisation :

**Voici une visualisation des variables qui montrent la dispersion des données autour de la moyenne.**



**Températures (T2M, T2M\_MAX, T2M\_MIN) :** Les distributions des températures sont concentrées autour des moyennes, avec quelques valeurs extrêmes, notamment pour T2M\_MAX, suggérant des pics de température.

**Humidité du Point de Rosée (T2MDEW) :** La distribution présente une légère asymétrie avec des valeurs plus fréquentes autour de 15-20°C, cohérente avec un climat chaud et humide.

**Pression (PS) :** La pression atmosphérique est relativement concentrée autour de 95-98 kPa, sans grandes fluctuations.

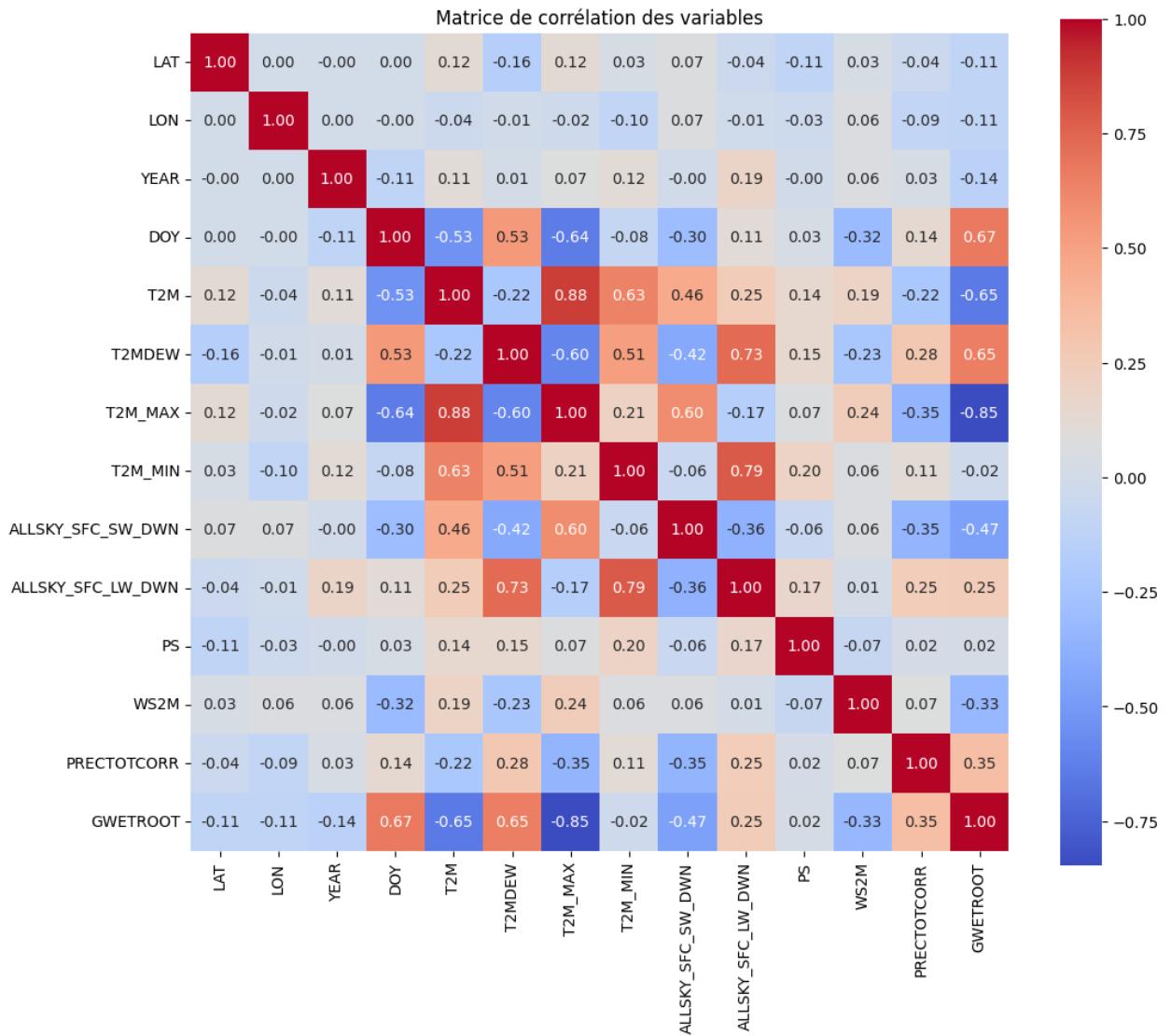
**Précipitations (PRECTOTCORR) :** La distribution est très asymétrique avec une majorité de valeurs faibles, et quelques valeurs très élevées (jusqu'à 848 mm), typiques des événements de fortes pluies.

**Humidité du Sol (GWETROOT) :** Les valeurs sont bien réparties entre 0.4 et 1, reflétant des niveaux variables d'humidité.

#### Corrélation entre variable :

Déterminons les relations entre les variables, par exemple, comment la température maximale est influencée par la radiation solaire ou la pression.

Utilisons une matrice de corrélation et une heatmap pour visualiser les corrélations entre toutes les variables.



### Corrélations fortes :

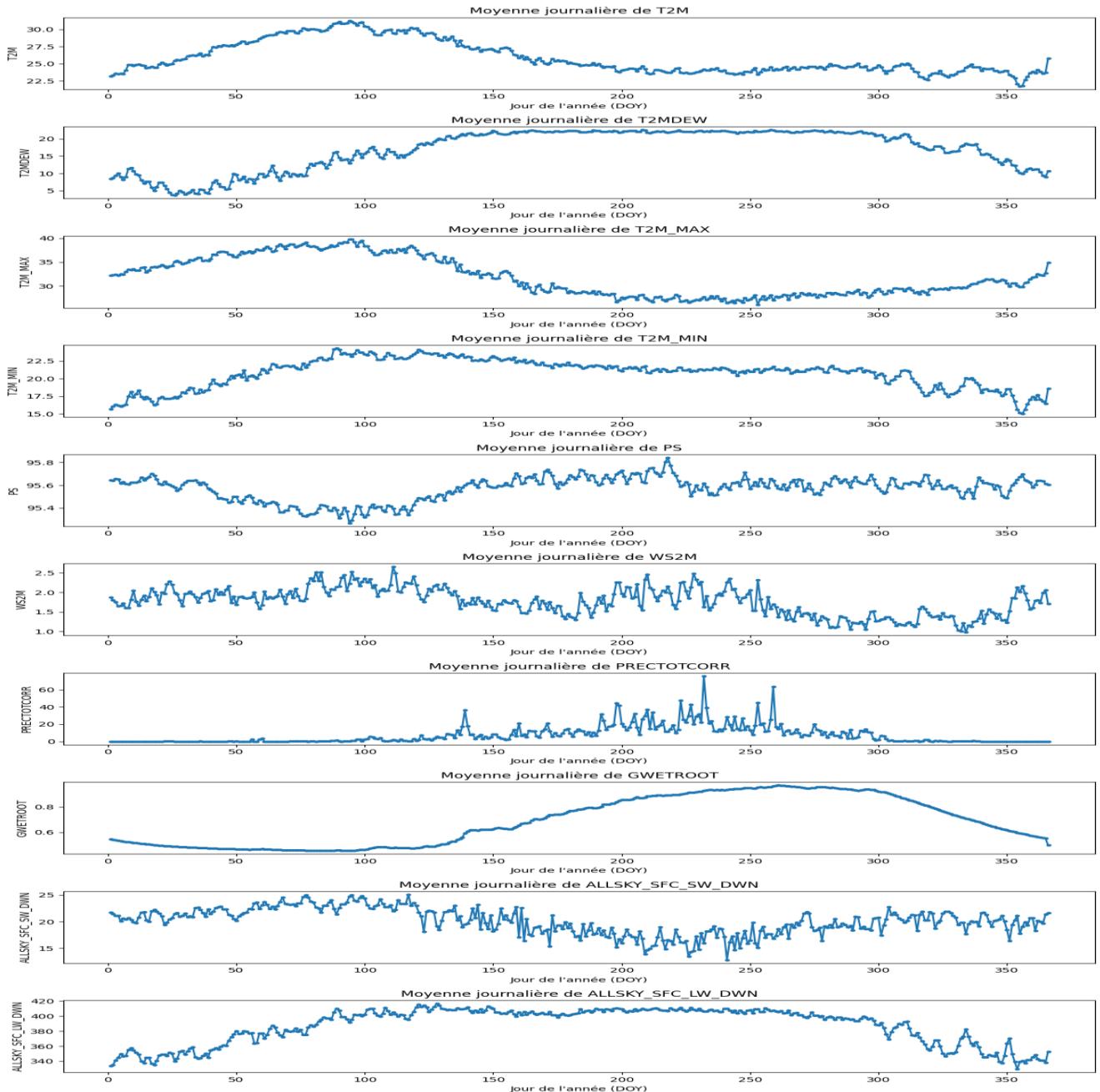
T2M (température moyenne) est fortement corrélée avec T2M\_MAX (température maximale) et T2M\_MIN (température minimale), ce qui est attendu car elles mesurent des aspects similaires de la température. T2MDEW (point de rosée) présente une corrélation élevée avec T2M, indiquant une relation entre la température et le niveau d'humidité. DOY (jour de l'année) est modérément corrélé avec GWETROOT (humidité du sol), suggérant une possible variation saisonnière de l'humidité du sol.

### Autres relations intéressantes :

ALLSKY\_SFC\_LW\_DWN (radiation descendante de longueurs d'onde) montre une forte corrélation avec T2M et T2MDEW, probablement liée à l'effet de la radiation sur la température. PRECTOTCORR (précipitations totales) a des corrélations faibles avec la plupart des autres variables, ce qui est fréquent car les précipitations peuvent être influencées par des facteurs externes non mesurés ici.

Compréhension de la tendance des données par jour, mois année et par saison :

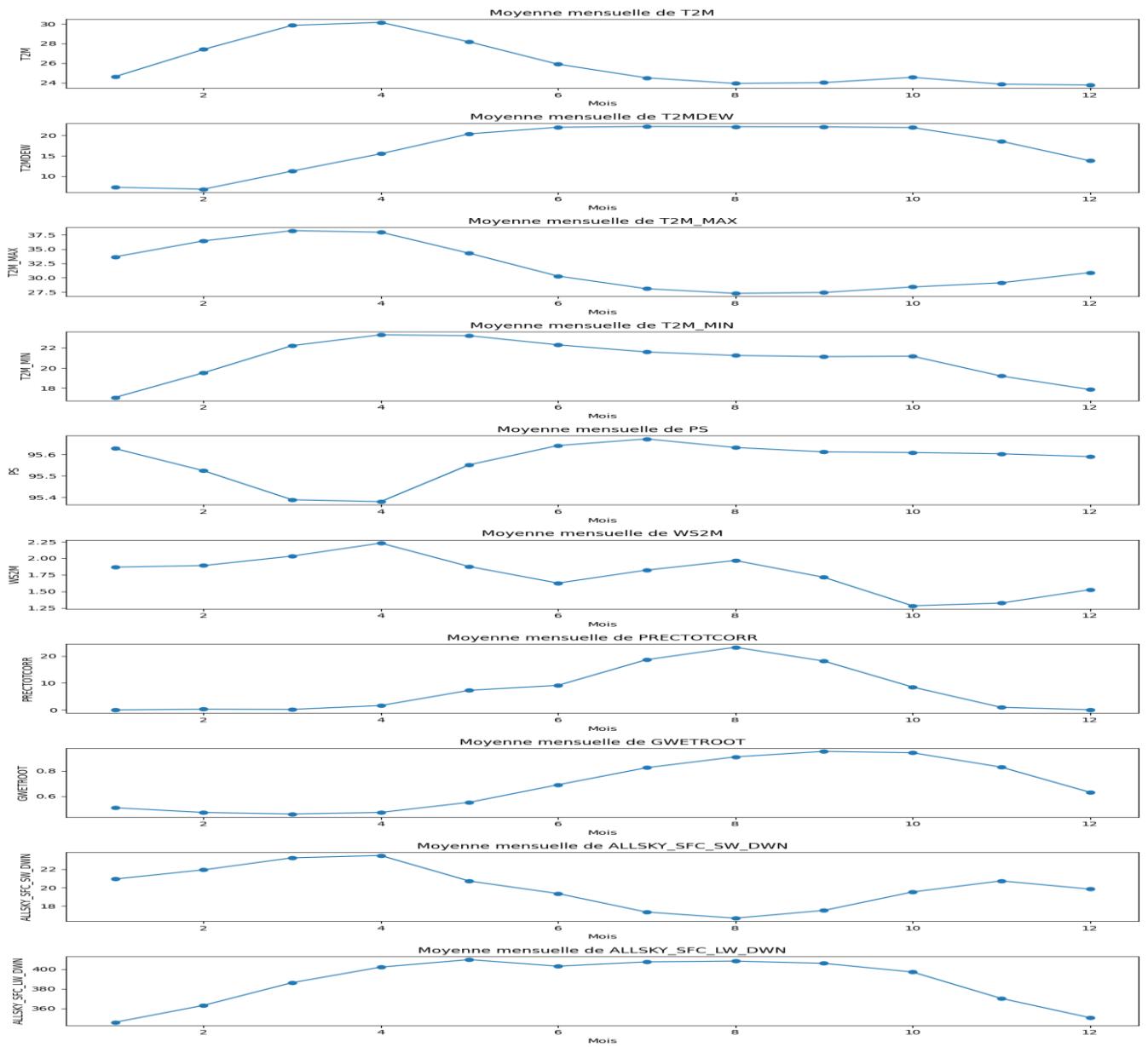
Par jour :



Voici les moyennes journalières pour chaque variable sur l'ensemble de l'année (par jour de l'année, DOY). Ces graphiques permettent d'observer en détail les fluctuations quotidiennes de chaque variable :

Températures et point de rosée : On peut voir des changements progressifs, révélant une variabilité plus fine que les moyennes mensuelles. Radiations solaires et précipitations : Les tendances journalières montrent plus de variabilité, en particulier dans la radiation solaire qui pourrait fluctuer en raison de la couverture nuageuse. Vitesse du vent et pression : Affichent également des variations journalières qui suivent un schéma plus régulier.

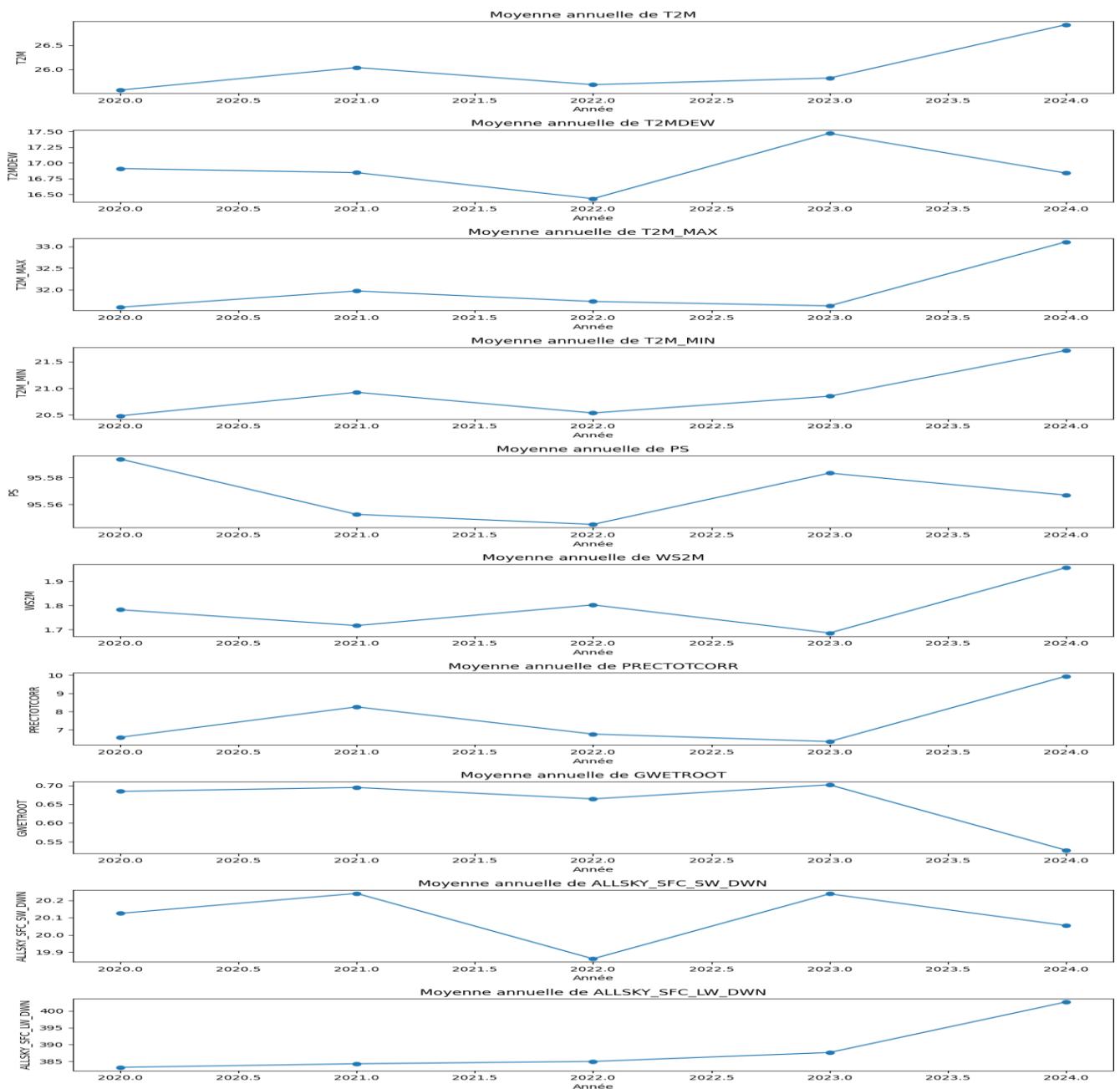
Par mois :



voici les moyennes mensuelles pour toutes les variables du jeu de données. Les graphes montrent les variations de chaque variable au cours des mois, permettant d'observer des tendances saisonnières :

Température (T2M, T2M\_MAX, T2M\_MIN) : Des variations claires avec des températures plus élevées au milieu de l'année. Point de rosée (T2MDEW) et Humidité du sol (GWETROOT) : Augmentation durant la saison des pluies. Radiations solaires (ALLSKY\_SFC\_SW\_DWN, ALLSKY\_SFC\_LW\_DWN) : Diminution en milieu d'année, possiblement due à la couverture nuageuse pendant la saison des pluies. Précipitations (PRECTOTCORR) : Pic pendant la saison des pluies (mois d'été). Vitesse du vent (WS2M) et Pression de surface (PS) : Présentent également des variations saisonnières légères

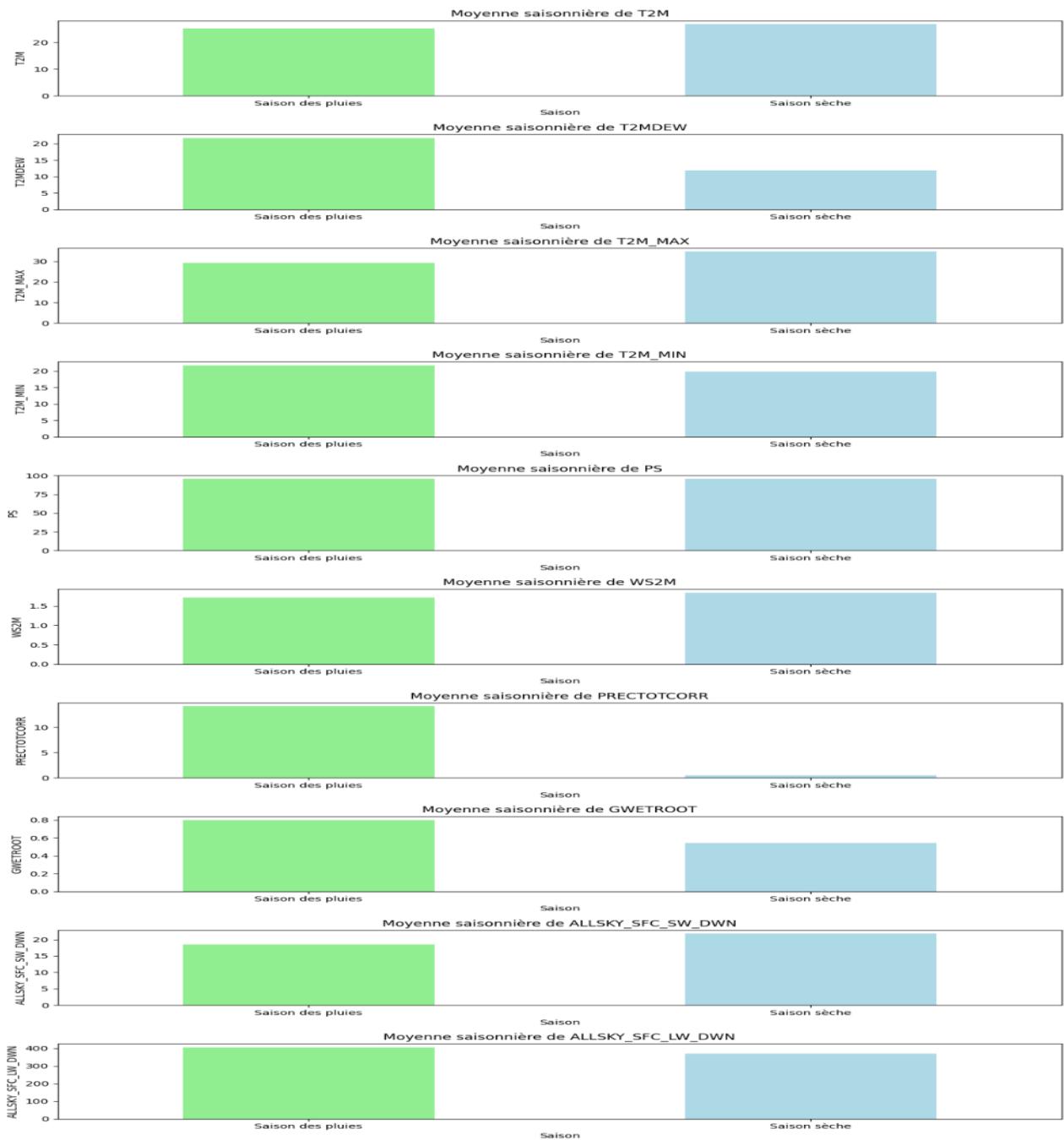
Par Année :



Voici les moyennes annuelles pour chaque variable du jeu de données. Ces graphiques montrent les tendances au fil des ans pour chaque variable :

Températures (T2M, T2M\_MAX, T2M\_MIN) : Les variations sont cohérentes d'année en année, mais certaines années peuvent être légèrement plus chaudes ou plus fraîches. Précipitations (PRECTOTCORR) : Les précipitations peuvent montrer des différences d'une année à l'autre, ce qui pourrait indiquer des variations climatiques saisonnières. Radiation solaire et pression de surface (ALLSKY\_SFC\_SW\_DWN, PS) :

Par saison :



En Moyenne Guinée :

- **Saison sèche** : Plus chaude et stable, avec moins de nuages, ce qui entraîne une radiation solaire plus forte et une pression de surface plus élevée.
- **Saison des pluies** : Plus fraîche, avec une humidité et des précipitations importantes, un point de rosée élevé, et une légère augmentation de la vitesse du vent.

## Préparation des données

Importation des données et organisation :

```
▶ #Librairie
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.impute import KNNImputer
from sklearn.preprocessing import *
from sklearn.compose import *

[ ] #chargons les données des 5 années de 2020/01/01 à 2024/09/30
data1=pd.read_csv("/content/1_POWER_Regional_Daily_20240101_20240930.csv",na_values=[-999])
data2=pd.read_csv("/content/2_POWER_Regional_Daily_20230101_20231231.csv",na_values=[-999])
data3=pd.read_csv("/content/3_POWER_Regional_Daily_20220101_20221201.csv",na_values=[-999])
data4=pd.read_csv("/content/4_POWER_Regional_Daily_20210101_20211231.csv",na_values=[-999])
data5=pd.read_csv("/content/5_POWER_Regional_Daily_20200101_20201231.csv",na_values=[-999])
```

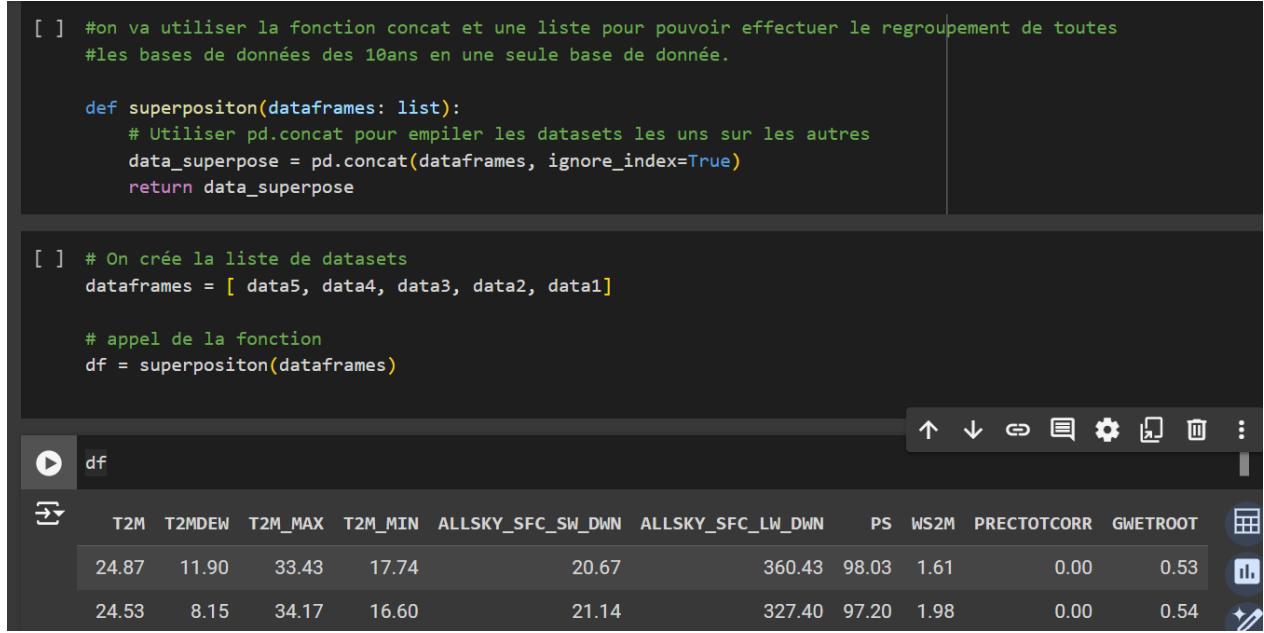
Après avoir importé les librairies, nous avons spécifié en chargeant les données : na\_values[-999] pour transformer toute colonne avec cette donnée en NaN. Ceci nous aidera devant pour reconnaître les données manquantes avec la fonction df.isnull().sum().

```
[ ] #on va utiliser la fonction concat et une liste pour pouvoir effectuer le regroupement de toutes
#les bases de données des 10ans en une seule base de donnée.

def superpositon(dataframes: list):
    # Utiliser pd.concat pour empiler les datasets les uns sur les autres
    data_superpose = pd.concat(dataframes, ignore_index=True)
    return data_superpose

[ ] # On crée la liste de datasets
dataframes = [ data5, data4, data3, data2, data1]

# appel de la fonction
df = superpositon(dataframes)
```



Nous avions plusieurs bases de données. Chaque base de données contenait les données pour une année et comme nous cherchons à travailler avec une seule base de données, on a fait la superposition des données par ordre croissant de 2020 à 2024.

Traitement appliqué pour la visualisation expliqué dans la partie compréhension des données :

Par jour :

```
df_doy = df[['DOY', 'T2M', 'T2MDEW', 'T2M_MAX', 'T2M_MIN', 'PS',
             'WS2M', 'PRECTOTCORR', 'GWETROOT', 'ALLSKY_SFC_SW_DWN', 'ALLSKY_SFC_LW_DWN']]
# Calculer les moyennes journalières pour toutes les variables
daily_trends_all = df_doy.groupby('DOY').mean()
```

Nous avons tout d'abord essayé d'extraire les variables qui nous intéressent et on a groupé les données par la moyenne des jours avec la fonction groupby() et mean() de cette façon on a pu obtenir la tendance basée sur le jour.

Par mois :

```
# Add a 'Month' column based on DOY for grouping
df['Date'] = pd.to_datetime(df['YEAR'] * 1000 + df['DOY'], format='%Y%j')
df['Month'] = df['Date'].dt.month
df_Month = df[['Month', 'T2M', 'T2MDEW', 'T2M_MAX', 'T2M_MIN',
               'PS', 'WS2M', 'PRECTOTCORR', 'GWETROOT', 'ALLSKY_SFC_SW_DWN', 'ALLSKY_SFC_LW_DWN']]

# Calculer les moyennes mensuelles pour toutes les variables disponibles
monthly_trends_all = df_Month.groupby('Month').mean()
```

Nous avons commencé par ajouté à notre dataset la variable mois en fonction de la variable jour et année. Puis nous avons regroupé les données en fonction de la moyenne mensuelle.

Par an :

```
df_year = df[['YEAR', 'T2M', 'T2MDEW', 'T2M_MAX', 'T2M_MIN',
              'PS', 'WS2M', 'PRECTOTCORR', 'GWETROOT', 'ALLSKY_SFC_SW_DWN', 'ALLSKY_SFC_LW_DWN']]
# Calculer les moyennes annuelles pour toutes les variables
annual_trends_all = df_year.groupby('YEAR').mean()
variablesA = annual_trends_all.columns
```

Vu que nous avons déjà la variable YEAR nous avons juste appliqué le même principe que pour le jour.

Par saison :

```
#créons la variable saison
def assign_season_by_month(month):
    if 5 <= month <= 10:
        return 'Saison des pluies'
    else:
        return 'Saison sèche'

# Appliquer la fonction pour créer la colonne 'Saison' en utilisant 'Month'
df['Saison'] = df['Month'].apply(assign_season_by_month)

numeric_columns = ['T2M', 'T2MDEW', 'T2M_MAX', 'T2M_MIN', 'PS', 'WS2M',
                   'PRECTOTCORR', 'GWETROOT', 'ALLSKY_SFC_SW_DWN', 'ALLSKY_SFC_LW_DWN']

# Calculer les moyennes par saison pour chaque variable numérique
seasonal_trends_all = df.groupby('Saison')[numeric_columns].mean()
```

Comme abordé précédemment dans le chapitre compréhension du domaine, la Guinée est un pays avec 2 saisons (sèche et pluvieuse) de mai à octobre pour la saison des pluies et de novembre à avril pour la

saison sèche. Nous avons tenu compte de cela pour les saisons et nous avons évaluer les tendances saisonnières.

**Doublons** : il est important de vérifier qu'il n'existe pas des doublons dans notre dataset. Avec la fonction `duplicated()` et `any()` on nous renverra un booléen : true s'il existe des doublons ou false pour le contraire.

```
✓ 0s ➜ has_duplicates = df.duplicated().any()
    print("Présence de lignes dupliquées :", has_duplicates)

➜ Présence de lignes dupliquées : False
```

Effectivement, nous n'avons pas de doublons dans notre dataset.

#### Calcul de la variable Target IRR :

Comme discuté en haut on va calculer irr qui est notre variable cible, On va utiliser la méthode de Penman Montheit pour le calcul de l'évapotranspiration ET0. Puis nous allons calculer ETC en multipliant ET0 par kc, mais comme kc est à 1 nous allons juste considérer ET0. Nous allons aussi soustraire à ET0 l'humidité du sol, la variable GWETROOT nous donne en pourcentage sur 1 le niveau d'humidité donc on va le multiplier par ET0 qui désigne combien en mm d'eau la plante a besoin pour savoir combien d'eau la plante peut récupérer à partir de l'humidité du sol puis le soustraire à ET0 et à la précipitation. Ceci diminuera notre quantité d'Irrigation à effectuer en considérant les facteurs humidité et précipitation.

```
▶ def irr(df):
    # Variables:

    T2M = df['T2M'].values
    T2MDEW = df['T2MDEW'].values
    T2M_MAX = df['T2M_MAX'].values
    T2M_MIN = df['T2M_MIN'].values
    Rns = df['ALLSKY_SFC_SW_DWN'].values
    Rnl = df['ALLSKY_SFC_LW_DWN'].values
    patm = df['PS'].values
    u2 = df['WS2M'].values
    P = df['PRECTOTCORR'].values # Précipitations totales corrigées
    pourcentage_humidite_Sol = df['GWETROOT'].values
    kc=1
    # Calcul de l'humidité de l'air saturé
    es = (0.6108 * np.exp((17.27 * T2M_MAX) / (T2M_MAX + 237.3)) +
          0.6108 * np.exp((17.27 * T2M_MIN) / (T2M_MIN + 237.3))) ** 2

    # Calcul de la pression de vapeur actuelle
    ea = 0.6108 * np.exp((17.27 * T2MDEW) / (T2MDEW + 237.3))
```

```

# Constante psychrométrique
gamma = 0.665 * 10**-3 * patm

# Calcul du flux de rayonnement net
Rn = Rnl - Rns # Flux de rayonnement net

# Calcul de Δ
delta = (4098 * es) / (237.3 + T2M)**2

# Calcul de ET0
ET0 = (0.408 * delta * Rn + gamma * (900 / (T2M + 273)) * u2 * (es - ea)) / (delta + gamma * (1 + 0.34 * u2))
# ETC=ET0*kc et comme kc=1 donc ETC=ET0
humidite_sol=ET0*pourcentage_humidite_Sol
# Calcul de IRR
IRR = ET0 - P - humidite_sol

# Ajout IRR aonsu DataFrame
df['IRR'] = IRR
return df

```

IRR
45.329149
39.752264
38.283148
35.514478
32.625592

Ainsi nous obtenons notre variable cible sur la quelle on va se baser pour faire notre prédition.

Corrélation des variables par rapport à IRR :

```
#j'ai envie de savoir la corrélation des autres variables par rapport à IRR
df_num = df.select_dtypes(include=['number'])
corr_matrix = df_num.corr()
corr_matrix["IRR"].sort_values(ascending=False)
```

	IRR
IRR	1.000000
T2M_MAX	0.773758
T2M	0.618872
Saison_Saison_seche	0.563744
ALLSKY_SFC_SW_DWN	0.478336
WS2M	0.181589
YEAR	0.131598
LON	0.102839
LAT	0.096035
T2M_MIN	0.062413
index	0.018332
PS	-0.010378
ALLSKY_SFC_LW_DWN	-0.176438
Month	-0.519363
DOY	-0.519662
T2MDEW	-0.519695
Saison_Saison_pluies	-0.563744
PRECTOTCORR	-0.775993
GWETROOT	-0.849921

Les températures élevées (T2M\_MAX, T2M) et les saisons sèches sont des facteurs positivement associés à l'IRR, tandis que les précipitations, l'humidité du sol, et les saisons pluvieuses sont des facteurs négativement associés. Les autres variables ont un impact limité ou négligeable.

#### Normalisation des données :

La normalisation est utilisée pour mettre les données sur une même échelle, ce qui est particulièrement utile pour les algorithmes de machine Learning et l'analyse statistique. La normalisation garantit que toutes les variables sont traitées sur un pied d'égalité, ce qui conduit à des modèles plus performants et plus interprétables.

La fonction MinMaxScaler de scikit learn nous aide à faire cela.

```
# Normalisation des fonctionnalités uniquement
feature_scaler = MinMaxScaler()
df_features_scaled = feature_scaler.fit_transform(df.drop(columns=['IRR'])) # Supposons que 'IRR' est notre variable cible

# Normalisation de la cible uniquement
target_scaler = MinMaxScaler()
df_target_scaled = target_scaler.fit_transform(df[['IRR']]) # Notez le double crochet pour garder une seule colonne

# Conversion en DataFrame normalisé pour la cible et les features
df_scaled = pd.DataFrame(df_features_scaled, columns=df.drop(columns=['IRR']).columns)
df_scaled['IRR'] = df_target_scaled # Ajout de la colonne cible normalisée
```

Nous avons aussi la variable saison qu'on doit transformer. On va utiliser la fonction get\_dummies de pandas pour la rendre catégoriques :

Saison_Saison_pluies	Saison_Saison_seche
0.0	1.0
0.0	1.0
0.0	1.0
0.0	1.0
0.0	1.0

Voici un affichage de quoi ressemble les données de saison divisée en 2 saison : pluvieuse et sèche.

Split et gestions des valeurs manquantes ainsi que des outliers :

Vérifions-s'il y a des données manquantes :

```
df_Nan = df[['T2M', 'T2MDEW', 'T2M_MAX', 'T2M_MIN', 'PS', 'WS2M',  
# Vérifions si des valeurs manquantes existent  
df_Nan.isnull().sum()
```

	0
T2M	96
T2MDEW	96
T2M_MAX	96
T2M_MIN	96
PS	96
WS2M	96
PRECTOTCORR	96
GWETROOT	2912

Effectivement il y a des données manquantes et la colonne pour l'humidité du sol est celle qui contient le plus de valeurs manquantes. Par contre, ces données manquantes n'atteignent même pas 10% des données de cette variable.

Gestion par suppression :

```
[4]: df.dropna(inplace=True)  
  
df.isnull().sum()  


|      | 0 |
|------|---|
| LAT  | 0 |
| LON  | 0 |
| YEAR | 0 |


```

### Split des données :

```
# Division Train, Validation (Test)
train_size = int(len(df_scaled) * 0.8) # 80% pour l'entraînement
test = len(df_scaled) - train_size # 20% pour validation (test)

# Division des données en train et test
train_data = df_scaled.iloc[:train_size] # 80% des données pour l'entraînement
test_data = df_scaled.iloc[train_size:] # 20% des données pour le test (validation)
```

On a fait la division de nos données comme suite : 80 % pour les données d'entraînement et 20% pour les données de test on a utilisé cette méthode pour suivre une continuité au fil du temps. On se base sur le passé pour prédire le futur.

### Gestion des NaN par interpolation :

Sans avoir exécuté le code sur la première méthode de gestion des NaN.

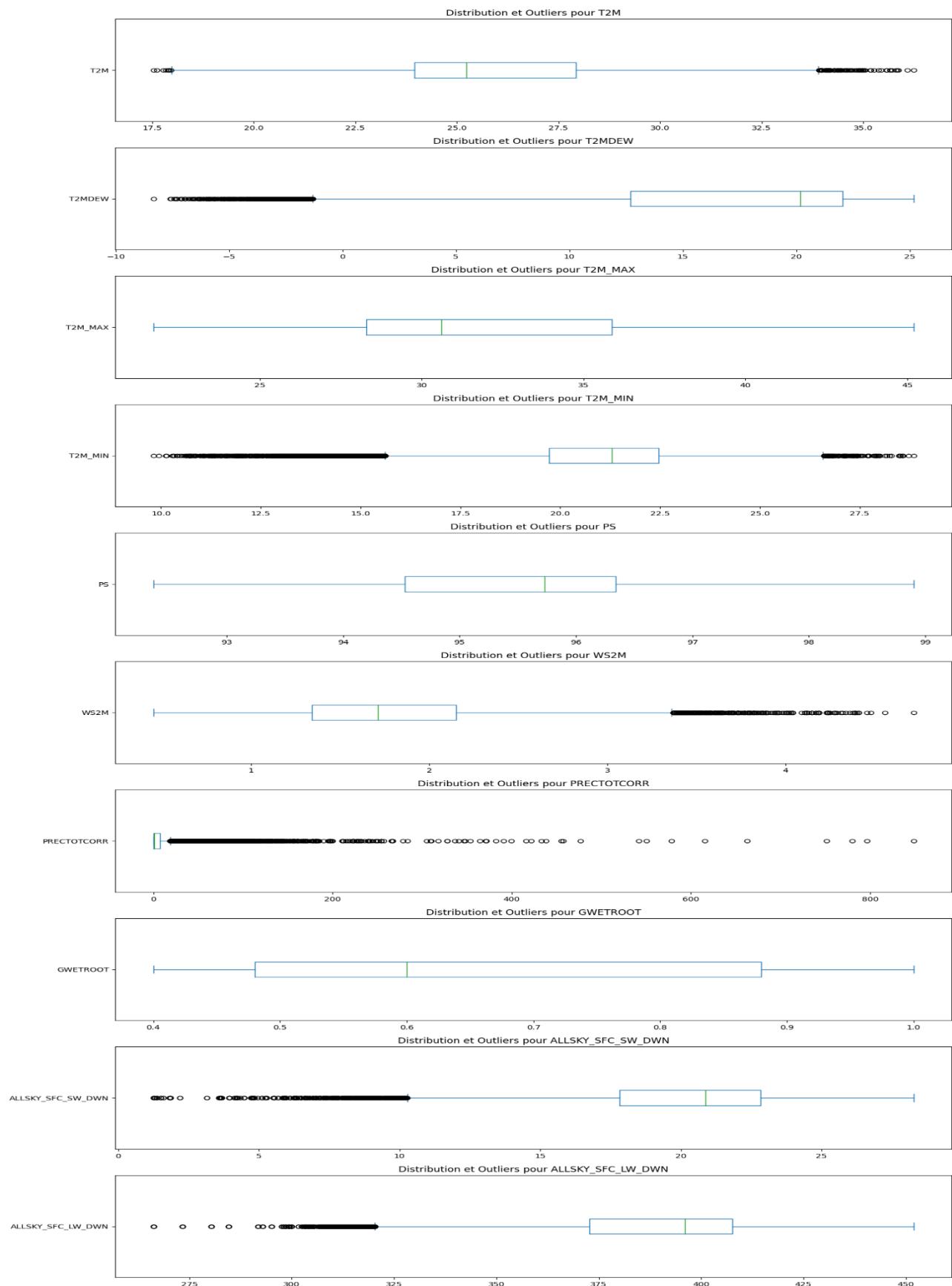
```
#####
#a executer uniquement si on doit gerer les nan
#####
# Interpolation for train_data
train_data=train_data.interpolate(method='linear', limit_direction='forward', axis=0)

# supprimons les nan de lensemble test
test_data=test_data.dropna()
#####
# Préparation des fonctionnalités (features) et des cibles (target)
```

On va tester les 2 résultats pour prendre une décision de la méthode à implémenté.

### Gestion des outliers :

Vérifions nos variables et analysons leurs pourcentages de données aberrantes avec une représentation en boxplot.



Il y'a beaucoup d'outliers testons plusieurs gestions de ces outliers par la moyenne, la médiane, l'interpolation et par campement en se basant sur la méthode des quartiles pour les intervalles.

```

def remove_outliers(x: pd.DataFrame) -> pd.DataFrame:
    cols = x.columns
    Q25 = x[cols].quantile(0.25) # Q1 (premier quartile)
    Q75 = x[cols].quantile(0.75) # Q3 (troisième quartile)
    IQR = Q75 - Q25
    SeuilMin = Q25 - 1.5 * IQR
    SeuilMax = Q75 + 1.5 * IQR
    # Filtrage des lignes sans outliers
    df_filtered = x.apply(lambda col: col[(col >= SeuilMin[col.name]) & (col <= SeuilMax[col.name])])
    return df_filtered.dropna()

def replace_outliers_with_mean(x: pd.DataFrame) -> pd.DataFrame:
    Q25 = x.quantile(0.25) # Q1
    Q75 = x.quantile(0.75) # Q3
    IQR = Q75 - Q25
    SeuilMin = Q25 - 1.5 * IQR
    SeuilMax = Q75 + 1.5 * IQR
    mean_values = x.mean()

    # Remplacement des outliers
    df_outliers_replaced = x.apply(
        lambda col: col.where((col >= SeuilMin[col.name]) & (col <= SeuilMax[col.name]), mean_values[col.name])
    )
    return df_outliers_replaced

median_values = x.median()

# Remplacement des outliers
df_outliers_replaced = x.apply(
    lambda col: col.where((col >= SeuilMin[col.name]) & (col <= SeuilMax[col.name]), median_values[col.name])
)
return df_outliers_replaced

# Marquer les outliers comme NaN
df_with_nans = x.apply(
    lambda col: col.where((col >= SeuilMin[col.name]) & (col <= SeuilMax[col.name]), np.nan)
)

# Remplacer les NaN par interpolation
df_outliers_replaced = df_with_nans.interpolate(method='linear', axis=0)

return df_outliers_replaced

```

```

# Fonction pour la méthode de campement
def apply_camping_method(data, lower_quantile=0.01, upper_quantile=0.99):

    lower_bound = np.quantile(data, lower_quantile, axis=0)
    upper_bound = np.quantile(data, upper_quantile, axis=0)
    return np.clip(data, lower_bound, upper_bound)

```

Ces méthodes sont appliquées que sur le dataset train et on a supprimé les outliers pour le data set du test.

Maintenant séparons les x et y pour avoir notre variable cible sur y.

```

# Préparation des fonctionnalités (features) et des cibles (target)
target_column = 'IRR' # Nom de la colonne cible
X_train = train_data.drop(columns=[target_column]) # Features d'entraînement (DataFrame Pandas)
y_train = train_data[[target_column]] # Cible d'entraînement (en DataFrame pour scaler)

X_test = test_data.drop(columns=[target_column]) # Features de test (DataFrame Pandas)
y_test = test_data[[target_column]] # Cible de test (en DataFrame pour scaler)

```

Ensute appliquons nos méthodes pour leur entraînement on va entraîner pour voir quelle méthode de gestion nous offre les meilleurs résultats.

```

# Appliquer la méthode de camping uniquement sur les données d'entraînement
X_train_camp = apply_camping_method(X_train, lower_quantile=0.01, upper_quantile=0.99)
y_train_camp = apply_camping_method(y_train, lower_quantile=0.01, upper_quantile=0.99)
X_train_moy = replace_outliers_with_mean(X_train)
y_train_moy = replace_outliers_with_mean(y_train)
X_train_median = replace_outliers_with_median(X_train)
y_train_median = replace_outliers_with_median(y_train)
X_train_inter = replace_outliers_with_interpolation(X_train)
y_train_inter = replace_outliers_with_interpolation(y_train)

```

Nous devons faire des reshape plus loin pour nos entraînements en deep learning et pour XGBoost.

```

# Reshape explicite pour les données d'entraînement
X_train_m = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)
y_train = y_train.values.reshape(-1, 1) # y_train en 2D si nécessaire

# Reshape explicite pour les données de test
X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)
y_test = y_test.values.reshape(-1, 1) # y_test en 2D si nécessaire

```

Car les modèles ont des standards de formatage de données qu'ils acceptent.

[Entrainement Modèles :](#)

Nous avons commencé par tester les quatre premières méthodes appliquées :

La gestion par la moyenne puis la médiane puis celle de l'Interpolation.

```
# Initialize the XGBRegressor with best hyperparameters
final_model = XGBRegressor(
    learning_rate=0.1,
    max_depth=7,
    n_estimators=150,
    objective='reg:squarederror' # Objective for regression
)

final_model.fit(X_train_moy, y_train_moy)

train_predictions = final_model.predict(X_train_moy)

# Calculate regression metrics
rmse = mean_squared_error(y_train_moy, train_predictions, squared=False)
mae = mean_absolute_error(y_train_moy, train_predictions)
mse = mean_squared_error(y_train_moy, train_predictions)
r2 = r2_score(y_train_moy, train_predictions)
```

**XGBRegressor** est une classe de la bibliothèque XGBoost qui implémente le boosting par gradient pour les tâches de régression.

Le modèle est configuré avec des hyperparamètres spécifiques pour optimiser ses performances.

Explication du code :

Configurons le modèle avec ces paramètres :

`learning_rate=0.1` : La vitesse à laquelle le modèle apprend.

`max_depth=7` : La profondeur maximale des arbres (pour capturer des relations complexes).

`n_estimators=150` : Le nombre d'arbres de décision utilisés.

`objective='reg:squarederror'` : Spécifie qu'on veut faire de la régression (prédire des nombres).

La fonction `fit ()` entraîne le modèle en utilisant :

`X_train_inter` : Les données explicatives (features).

`Y_train_inter` : Les valeurs qu'on cherche à prédire.

Pendant l'entraînement, le modèle apprend à corriger ses erreurs.

La fonction `predict ()` donne les prédictions du modèle sur les données d'entraînement (`X_train_inter`).

Ces prédictions sont stockées dans `train_predictions`.

**Maintenant testons notre modèle sur les données de test.**

```
[60] # Predict on the test set
test_predictions = final_model.predict(X_test)

# Calculate regression metrics
rmse = mean_squared_error(y_test, test_predictions, squared=False)
mae = mean_absolute_error(y_test, test_predictions)
mse = mean_squared_error(y_test, test_predictions)
r2 = r2_score(y_test, test_predictions)

print("test RMSE:", rmse)
print("test MAE:", mae)
print("test MSE:", mse)
print("test R2 Score:", r2)
```

Nous avons procédé de cette façon pour les autres méthodes.

Voici les résultats que nous avons obtenu à l'entraînement et ceux obtenus au test :

Avec la suppression des données manquantes :

Méthode	RMSE		MAE		MSE		R CARRÉ		performance
train/test	train	test	train	test	train	test	train	test	
Campement	0.0003	0.08	0.0002	0.0007	1.44	7.09	0.99	0.93	Meilleure
Interpolation	0.003	0.01	0.001	0.003	1.37	0.0001	0.98	0.84	
Médiiane	0.004	0.013	0.001	0.002	2.17	0.0001	0.97	0.83	
Moyenne	0.004	0.01	0.001	0.002	1.88	0.0001	0.97	0.84	

Avec la gestion des données manquantes :

Méthode	RMSE		MAE		MSE		R CARRÉ		performance
train/test	train	test	train	test	train	test	train	test	
Campement	0.0003	0.001	0.0002	0.0005	1.51	1.75	0.99	0.99	Meilleure
Interpolation	0.005	0.0003	0.001	0.0002	3.23	1.51	0.93	0.99	
Médiiane	0.004	0.005	0.001	0.001	2.04	3.32	0.97	0.93	
Moyenne	0.004	0.005	0.001	0.001	1.98	3.17	0.97	0.94	

**Le meilleur modèle est celui avec la gestion des données manquantes avec campement.**

Explication du modèle choisi et pourquoi :

La fonction `apply_camping_method` gère les valeurs aberrantes dans un dataset en limitant les données à une plage définie par des quantiles (par défaut, 1% et 99%).

Objectif : Réduire l'impact des valeurs extrêmes (outliers) en les "clippant" (les ramener à des bornes définies). Voici les étapes expliquées :

Calcule le quantile inférieur (lower\_bound) et supérieur (upper\_bound).

Remplace les valeurs inférieures à lower\_bound par cette borne.

Remplace les valeurs supérieures à upper\_bound par cette borne.

Avantages :

Diminue l'effet des outliers sans exclure de données.

Stabilise les modèles sensibles aux valeurs extrêmes (régressions, réseaux neuronaux, etc.).

Nous voulons limiter l'impact des outliers sans les supprimer ni les gérer avec la moyenne ou encore la médiane car dans un contexte réel il peut y avoir de forte pluie dans la saison pluvieuse et de forte chaleur dans la saison sèche d'où nous allons considérer la méthode de gestion des outliers par camping

**Maintenant que nous avons la méthode qu'on va utiliser pour gérer les outliers, commençons l'entraînement de nos modèles.**

**Nous allons effectuer la validation croisée avec différents plis : 5, 7 et 10.**

**XGBoost regressor**

**Effectuons le gridsearch pour trouver les meilleurs paramètres pour chaque split :**

```
in # Configuration de TimeSeriesSplit
n_splits = 5
tscv = TimeSeriesSplit(n_splits=n_splits)

# Définir la grille d'hyperparamètres
param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 150, 200]
}

# Définir le modèle XGBoost
model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Configuration de GridSearchCV
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    scoring='neg_mean_squared_error', # Utiliser MSE comme métrique d'évaluation
    cv=tscv,
    verbose=2,
    n_jobs=-1
)
```

```

# Ajuster GridSearchCV sur les données d'entraînement
grid_search.fit(X_train_reshaped, y_train_reshaped)

# Résultats des meilleurs hyperparamètres
best_params = grid_search.best_params_
best_score = -grid_search.best_score_ # MSE (le score est retourné en négatif)

print("\nMeilleurs hyperparamètres trouvés par GridSearchCV:")
print(best_params)
print(f"Meilleur score (MSE): {best_score}")

# Résultats détaillés de GridSearchCV
cv_results = grid_search.cv_results_

```

**Split=5**

**On a eu ces meilleurs paramètres**

```

Meilleurs hyperparamètres trouvés par GridSearchCV:
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
.
```

**Split=7**

**On a eu ces meilleurs paramètres**

```

Meilleurs hyperparamètres trouvés par GridSearchCV:
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200}
.
```

**Split=10**

```

Meilleurs hyperparamètres trouvés par GridSearchCV:
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
.
```

**On a eu ces meilleurs paramètres**

Sur ce modèle nous effectuons pour le split 5, les modèles ont été entraînés avec les mêmes paramètres que nous avons vus précédemment et nous avons juste modifié au niveau du n\_splits.

```

# Configuration de TimeSeriesSplit
n_splits = 5 # Nombre de divisions pour la validation croisée
tscv = TimeSeriesSplit(n_splits=n_splits)

# Liste pour stocker les scores des métriques
mse_scores = []
mae_scores = []
r2_scores = []
rmse_scores = []

# Mesure des temps pour la validation croisée
validation_times = []

```

Tout d'abord on commence par initier nos variables. On fera aussi la validation croisée avec le timeSeriesSplit il est spécialement conçu pour les séries temporelles.

```

split_num = 1
for train_idx, val_idx in tscv.split(X_train_reshaped):
    print(f"\nProcessing Split {split_num}/{n_splits}")
    split_start_time = time.time() # Démarrer le chronomètre pour ce split

    # Division des données en train et validation pour ce split
    X_train_split, X_val_split = X_train_reshaped[train_idx], X_train_reshaped[val_idx]
    y_train_split, y_val_split = y_train_reshaped[train_idx], y_train_reshaped[val_idx]

    # Définition du modèle XGBoost Regressor
    model = XGBRegressor(
        learning_rate=0.1,
        max_depth=5,
        n_estimators=200,
        objective='reg:squarederror'
    )

    # Entraînement du modèle
    model.fit(X_train_split, y_train_split)

```

On configure une boucle qui va appliquer l'entraînement selon le split choisi avec les paramètres déjà évoqué ultérieurement.

Ci-dessous le modèle test sur les données de validation su split sur lequel il s'entraîne afin d'apprendre de ses erreurs.

```

# Prédictions sur le jeu de validation
y_val_pred = model.predict(X_val_split)

# Re-inverse pour ramener les prédictions et les valeurs réelles à l'échelle d'origine
y_val_pred_original = target_scaler.inverse_transform(y_val_pred.reshape(-1, 1))
y_val_original = target_scaler.inverse_transform(y_val_split.reshape(-1, 1))

# Calcul des métriques sur l'échelle originale
mse = mean_squared_error(y_val_original, y_val_pred_original)
mae = mean_absolute_error(y_val_original, y_val_pred_original)
rmse = np.sqrt(mse)
r2 = r2_score(y_val_original, y_val_pred_original)

# Stockage des scores
mse_scores.append(mse)
mae_scores.append(mae)
rmse_scores.append(rmse)
r2_scores.append(r2)

```

Vu que nos données sont normalisées, on veut effectuer le processus inverse pour nous faciliter l'interprétation des résultats.

Puis nous calculons les métriques et les stockons.

```

# Calcul du temps pour ce split
split_time = time.time() - split_start_time
validation_times.append(split_time)
print(f"Split {split_num} Time: {int(split_time // 60)} min {int(split_time % 60)} sec")
print(f"Split Metrics - MSE: {mse}, MAE: {mae}, RMSE: {rmse}, R²: {r2}")

split_num += 1

# Moyennes des métriques sur tous les splits
print("\nCross-Validation Metrics (Original Scale):")
print(f"Average MSE: {np.mean(mse_scores)}")
print(f"Average MAE: {np.mean(mae_scores)}")
print(f"Average RMSE: {np.mean(rmse_scores)}")
print(f"Average R²: {np.mean(r2_scores)}")

# Temps total pour la validation croisée
total_validation_time = sum(validation_times)
print(f"\nTotal Validation Time: {int(total_validation_time // 60)} min {int(total_validation_time % 60)} sec"

# Évaluation finale sur le jeu de test
print("\nTesting on the Test Set...")
test_start_time = time.time() # Démarrer le chronomètre pour le test

```

Nous avons lancé une minuterie qui nous affichera le temps que fera la validation croisée ainsi que le test.

```

# Prédiction sur le jeu de test
y_test_pred = model.predict(X_test_reshaped)
model.save_model('xgboost_model_5.json')

# Re-inverse pour ramener les prédictions et les valeurs réelles à l'échelle d'origine
y_test_pred_original = target_scaler.inverse_transform(y_test_pred.reshape(-1, 1))
y_test_original = target_scaler.inverse_transform(y_test_reshaped.reshape(-1, 1))

# Calcul des métriques sur le jeu de test
mse_test = mean_squared_error(y_test_original, y_test_pred_original)
mae_test = mean_absolute_error(y_test_original, y_test_pred_original)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test_original, y_test_pred_original)

# Calcul du temps pour le test
test_time = time.time() - test_start_time
print(f"\nTest Time: {int(test_time // 60)} min {int(test_time % 60)} sec")

# Affichage des métriques sur le jeu de test
print("\nTest Set Metrics (Original Scale):")
print(f"MSE: {mse_test}")
print(f"MAE: {mae_test}")
print(f"RMSE: {rmse_test}")
print(f"R²: {r2_test}")

```

Une fois la validation achevée on test sur l'ensemble test et on affiche les métriques.

Il faut noter aussi qu'on a enregistrer le modèle pour qu'on puisse le déployer facilement.

```

# Affichage des métriques sur le jeu de test
print("\nTest Set Metrics (Original Scale):")
print(f"MSE: {mse_test}")
print(f"MAE: {mae_test}")
print(f"RMSE: {rmse_test}")
print(f"R²: {r2_test}")

# Visualisation des résultats sur le jeu de test
plt.figure(figsize=(10, 6))
plt.plot(y_test_original[:100], label="Valeurs Réelles", alpha=0.8) # Les 100 premières valeurs pour plus de
plt.plot(y_test_pred_original[:100], label="Valeurs Prédites", alpha=0.8)
plt.title("Jeu de Test : Valeurs Réelles vs Prédiction (Échelle Originale)")
plt.xlabel("Index des Échantillons")
plt.ylabel("Valeur Cible (Échelle Originale)")
plt.legend()
plt.show()

```

On a configuré une visualisation qui va nous montrer les performances de notre modèle ceci nous facilitera l'interprétation et la validation du modèle.

## Model Deep Learning:

Nous avons entraîné plusieurs modèles de deep learning qui sont : LSTM, BI-LSTM, RNN ET GRU

**Deep Learning** : Branche de l'IA qui utilise des réseaux de neurones pour apprendre à partir de grandes données, souvent pour les séries temporelles, les images ou le texte.

**LSTM** : Réseau spécialisé pour les données séquentielles, il retient les informations sur le long terme.

**BI-LSTM** : Version avancée de LSTM qui regarde à la fois le passé et le futur pour mieux comprendre le contexte.

**RNN** : Réseau de base pour les séquences, mais limité pour les longues dépendances.

**GRU** : Similaire à LSTM, mais plus rapide et plus simple.

**Commençons par LSTM :**

**On va chercher à optimiser les paramètres pour chaque pli.**

```
# Fonction objectif pour Optuna
def objective(trial):
    # Définir l'espace de recherche pour les hyperparamètres
    n_units = trial.suggest_int('n_units', 50, 150, step=50) # 50 à 150 neurones par pas de 50
    dropout_rate = trial.suggest_float('dropout_rate', 0.1, 0.3, step=0.1) # Dropout entre 0.1 et 0.3
    learning_rate = trial.suggest_float('learning_rate', 0.001, 0.01, log=True)
    batch_size = trial.suggest_categorical('batch_size', [32, 64]) # Choix de 32 ou 64

# Compiler le modèle
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='mse')

# Étudier avec Optuna
study = optuna.create_study(direction='minimize') # Minimiser le MSE
study.optimize(objective, n_trials=5)
```

On a fourni un nombre de paramètres et on a fais l'optimisation bayésienne en utilisant optuna pour tuner nos paramètres.

Une fois qu'on a trouvé nos meilleurs paramètres on entraîne nos modèles en effectuant une validation croisée puis on le test avec notre ensemble de données de test.

```

min ⏎ # Charger les scalers pour normalisation et re-inverse
target_scaler = joblib.load('target_scaler.pkl') # Scaler utilisé pour la cible

# Configuration de TimeSeriesSplit
n_splits = 5 # Nombre de divisions pour la validation croisée
tscv = TimeSeriesSplit(n_splits=n_splits)

# Meilleurs paramètres
best_params = {'n_units': 150, 'dropout_rate': 0.2, 'learning_rate': 0.0013596813107823818, 'batch_size': 32}

# Liste pour stocker les scores des métriques
mse_scores = []
mae_scores = []
r2_scores = []
rmse_scores = []

# Liste pour les temps d'exécution
validation_times = []

```

Comme récemment on a initialisé nos variables, on a défini nos best params et on a utilisé timeSeriesSplit pour la validation croisée.

```

# Parcours des splits de la validation croisée
split_num = 1
for train_idx, val_idx in tscv.split(X_train):
    print(f"\nProcessing Split {split_num}/{n_splits}")
    split_start = time() # Démarrer le chronomètre pour ce split

    # Division des données en train et validation pour ce split
    X_train_split, X_val_split = X_train[train_idx], X_train[val_idx]
    y_train_split, y_val_split = y_train[train_idx], y_train[val_idx]

    # Définition du modèle LSTM avec les meilleures paramètres
    model = Sequential()
    model.add(LSTM(best_params['n_units'], activation='relu', return_sequences=True,
                   input_shape=(X_train_split.shape[1], X_train_split.shape[2])))
    model.add(Dropout(best_params['dropout_rate'])) # Régularisation
    model.add(LSTM(best_params['n_units'], activation='relu'))
    model.add(Dropout(best_params['dropout_rate'])) # Régularisation
    model.add(Dense(1)) # Couche de sortie pour la régression

    # Compilation du modèle avec le meilleur learning rate
    optimizer = tf.keras.optimizers.Adam(learning_rate=best_params['learning_rate'])
    model.compile(optimizer=optimizer, loss='mse')

```

Puis nous avons défini la même boucle pour la validation croisée et nous avons utilisé Keras pour notre entraînement.

**Keras** est une bibliothèque haut niveau intégrée à TensorFlow qui simplifie la création et l'entraînement de modèles de deep learning.

**TensorFlow** est une bibliothèque plus large pour le calcul numérique et le machine learning, dont Keras fait partie pour construire et entraîner des modèles.

model = Sequential() : On crée un modèle simple où les couches s'ajoutent les unes après les autres.

model.add(LSTM(..., activation='relu', return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2]))) : On ajoute une couche LSTM avec ... neurones en fonction des best params.

relu : C'est l'activation qui aide à mieux apprendre.

return\_sequences=True : Permet de garder la suite des données pour la couche suivante.

model.add(Dropout(..)) : On enlève des neurones au hasard pendant l'entraînement pour éviter le surapprentissage.

model.add(LSTM(..., activation='relu')) :

Une deuxième couche LSTM avec des neurones, mais cette fois, elle ne garde pas les séquences, on veut qu'elle produise une sortie unique résumant toute l'information séquentielle capturée par les couches précédentes.

model.add(Dropout) : On refait un dropout pour mieux généraliser.

model.add(Dense(1)) : Une dernière couche pour donner une seule valeur en sortie (prédiction).

Pour résumé on construit un modèle avec 2 couches LSTM pour bien comprendre les données en séquence, des dropout pour éviter les erreurs, et une couche finale pour prédire un résultat.

```
# Entraînement du modèle
history = model.fit(X_train_split, y_train_split,
                     epochs=20, # Garder 20 époques
                     batch_size=best_params['batch_size'], # Meilleure taille de batch
                     validation_data=(X_val_split, y_val_split),
                     verbose=0)

# Prédictions sur le jeu de validation
y_val_pred = model.predict(X_val_split)
.

# Re-inverse pour ramener les prédictions et les valeurs réelles à l'échelle d'origine
y_val_pred_original = target_scaler.inverse_transform(y_val_pred)
y_val_original = target_scaler.inverse_transform(y_val_split)

# Calcul des métriques sur l'échelle originale
mse = mean_squared_error(y_val_original, y_val_pred_original)
mae = mean_absolute_error(y_val_original, y_val_pred_original)
rmse = np.sqrt(mse)
r2 = r2_score(y_val_original, y_val_pred_original)
```

Nous ajustons le modèle ces paramètres :

epochs=20 : Le modèle passe 20 fois sur l'ensemble d'entraînement.

batch\_size : Les données sont traitées par blocs à chaque étape.

validation\_data=(X\_val\_split , y\_val\_split) : Évalue les performances sur l'ensemble de validation à chaque époque.

verbose=1 : Affiche les détails de l'entraînement (perte, métriques, etc.).

Cette instruction au niveau de history= model.fit... Entraîne le modèle pendant 20 époques, en utilisant des blocs de 64 données, tout en suivant ses performances sur un ensemble de validation.

```
# Stockage des scores
mse_scores.append(mse)
mae_scores.append(mae)
rmse_scores.append(rmse)
r2_scores.append(r2)

# Calcul du temps d'exécution pour ce split
split_time = time() - split_start
validation_times.append(split_time)

print(f"Split {split_num} Time: {int(split_time // 60)} min {int(split_time % 60)} sec")
print(f"Split Metrics - MSE: {mse}, MAE: {mae}, RMSE: {rmse}, R²: {r2}")

split_num += 1

# Moyennes des métriques sur tous les splits
print("\nCross-Validation Metrics (Original Scale):")
print(f"Average MSE: {np.mean(mse_scores)}")
print(f"Average MAE: {np.mean(mae_scores)}")
print(f"Average RMSE: {np.mean(rmse_scores)}")
print(f"Average R²: {np.mean(r2_scores)}")

# Temps total pour la validation croisée
total_validation_time = sum(validation_times)
print(f"\nTotal Validation Time: {int(total_validation_time // 60)} min {int(total_validation_time % 60)} sec")

# Sauvegarde du modèle final
model.save('lstm5_best_params.keras')
```

Après l'entraînement les métriques sont calculés et les résultats affichées.

```
# Évaluation finale sur le jeu de test
print("\nTesting on the Test Set...")
test_start = time() # Démarrer le chronomètre pour le test

# Prédictions sur le jeu de test
y_test_pred = model.predict(X_test)

# Re-inverse pour ramener les prédictions et les valeurs réelles à l'échelle d'origine
y_test_pred_original = target_scaler.inverse_transform(y_test_pred)
y_test_original = target_scaler.inverse_transform(y_test)

# Calcul des métriques sur l'échelle originale
mse_test = mean_squared_error(y_test_original, y_test_pred_original)
mae_test = mean_absolute_error(y_test_original, y_test_pred_original)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test_original, y_test_pred_original)

# Calcul du temps d'exécution pour le test
test_time = time() - test_start
print(f"\nTest Time: {int(test_time // 60)} min {int(test_time % 60)} sec")
```

On aussi intégré une minuterie. Et on a tester sur les données de test.

```

# Affichage des métriques sur le jeu de test
print("\nTest Set Metrics (Original Scale):")
print(f"MSE: {mse_test}")
print(f"MAE: {mae_test}")
print(f"RMSE: {rmse_test}")
print(f"R²: {r2_test}")

# Visualisation des résultats sur le jeu de test
plt.figure(figsize=(10, 6))
plt.plot(y_test_original[:100], label="True Values", alpha=0.8) # Afficher les 100 premières valeurs réelles
plt.plot(y_test_pred_original[:100], label="Predicted Values", alpha=0.8)
plt.title("Test Set: True Values vs Predictions (Original Scale)")
plt.xlabel("Sample Index")
plt.ylabel("Target Value (Original Scale)")
plt.legend()
plt.show()

```

Et on a visualisé pour interpréter et valider notre modèle entraîné.

Voici la méthode utilisée pour entraîner le modèle LSTM et nous vous spécifions que tous les autres modèles de deep learning que nous avons choisie dans notre étude ont été entraîné avec les mêmes paramètres pour faire une évaluation équitable entre les modèles.

On fait de même pour les BI-LSTM, RNN et GRU juste en modifiant les modèles comme présenté ci-dessous.

```

# Définition du modèle LSTM avec les meilleurs paramètres
model = Sequential()
model.add(Bidirectional(LSTM(best_params['n_units'], activation='relu', return_sequences=True,
                             input_shape=(X_train_split.shape[1], X_train_split.shape[2]))))
model.add(Dropout(best_params['dropout_rate'])) # Régularisation
model.add(Bidirectional(LSTM(best_params['n_units'], activation='relu')))
model.add(Dropout(best_params['dropout_rate'])) # Régularisation
model.add(Dense(1)) # Couche de sortie pour la régression

```

```

# Définition du modèle LSTM avec les meilleurs paramètres
model = Sequential()
model.add(GRU(best_params['n_units'], activation='relu', return_sequences=True,
              input_shape=(X_train_split.shape[1], X_train_split.shape[2])))
model.add(Dropout(best_params['dropout_rate'])) # Régularisation
model.add(GRU(best_params['n_units'], activation='relu'))
model.add(Dropout(best_params['dropout_rate'])) # Régularisation
model.add(Dense(1)) # Couche de sortie pour la régression

```

```

# Définition du modèle LSTM avec les meilleurs paramètres
model = Sequential()
model.add(SimpleRNN(best_params['n_units'], activation='relu', return_sequences=True,
                    input_shape=(X_train_split.shape[1], X_train_split.shape[2])))
model.add(Dropout(best_params['dropout_rate'])) # Régularisation
model.add(SimpleRNN(best_params['n_units'], activation='relu'))
model.add(Dropout(best_params['dropout_rate'])) # Régularisation
model.add(Dense(1)) # Couche de sortie pour la régression

```

On eut ceci pour les best params :

### Pour LSTM

Plis 5-7-10 successivement

```

Best Parameters:
{'n_units': 150, 'dropout_rate': 0.2, 'learning_rate': 0.0013596813107823818, 'batch_size': 32}

Best Parameters:
{'n_units': 50, 'dropout_rate': 0.1, 'learning_rate': 0.0038752126427292997, 'batch_size': 32}

Best Parameters:
{'n_units': 100, 'dropout_rate': 0.1, 'learning_rate': 0.0034929331219063307, 'batch_size': 64}

```

### Pour bi lstm :

Plis 5-7-10 successivement

```

Best Parameters:
{'n_units': 100, 'dropout_rate': 0.3, 'learning_rate': 0.001957077778916276, 'batch_size': 64}

# Meilleurs paramètres
best_params = {'n_units': 150, 'dropout_rate': 0.3, 'learning_rate': 0.0013968728396077767, 'batch_size': 32}

Best Parameters:
{'n_units': 100, 'dropout_rate': 0.1, 'learning_rate': 0.00892958356475852, 'batch_size': 64}

```

### Pour Rnn :

Plis 5-7-10 successivement

```

Best Parameters:
{'n_units': 50, 'dropout_rate': 0.3, 'learning_rate': 0.002650624529478377, 'batch_size': 32}
Best MSE: 58.250660380375734

Best Parameters:
{'n_units': 50, 'dropout_rate': 0.1, 'learning_rate': 0.0012471925475110437, 'batch_size': 32}
Best MSE: 59.911479819228155

```

```
Best Parameters:
{'n_units': 150, 'dropout_rate': 0.1, 'learning_rate': 0.0014048950410855277, 'batch_size': 32}
```

Pour gru :

Plis 5-7-10 successivement

```
Best Parameters:
{'n_units': 150, 'dropout_rate': 0.2, 'learning_rate': 0.001459543696199874, 'batch_size': 32}
Best MSE: 93.25915495600478
```

```
Best Parameters:
{'n_units': 100, 'dropout_rate': 0.2, 'learning_rate': 0.0038797356114489596, 'batch_size': 32}
Best MSE: 126.80629064882191
```

```
Best Parameters:
{'n_units': 50, 'dropout_rate': 0.3, 'learning_rate': 0.002175864766695924, 'batch_size': 32}
Best MSE: 229.41026006700006
```

### Evaluation des modèles :

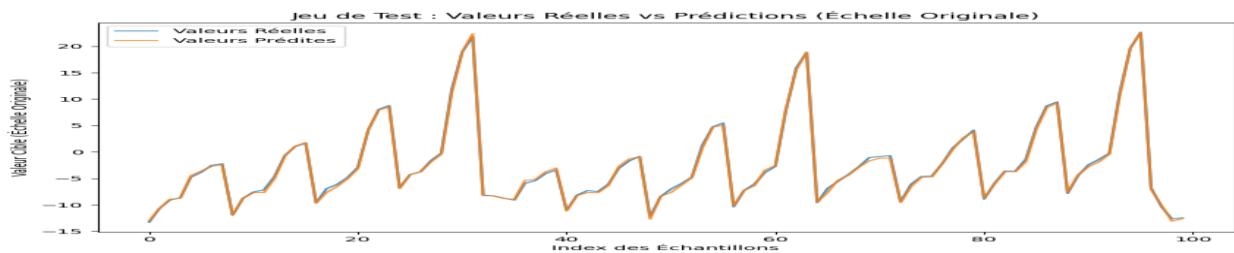
Nous allons faire l'évaluation de nos différents modèles

Présentation du tableau des résultats obtenus dans les différents plis pour XGBoost

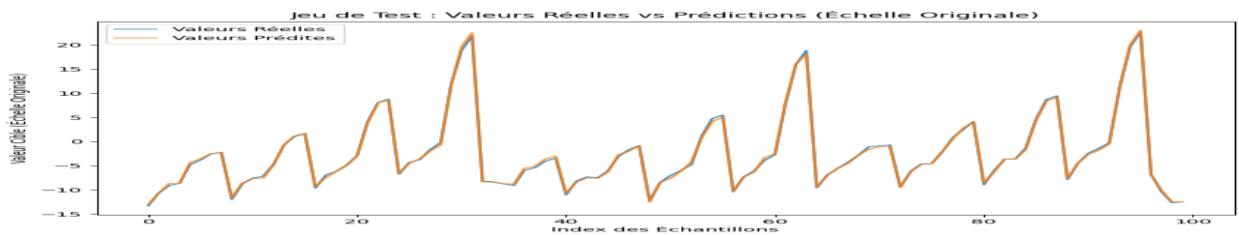
Méthode	RMSE		MAE		MSE		R CARRÉ		Performance
train/test	train	test	train	test	train	test	train	test	
Pli5	0.84	1.29	0.52	0.55	0.72	1.66	0.9989	0.9963	
Pli7	1.13	1.25	0.73	0.53	2.00	1.56	0.9952	0.9966	Meilleur
Pli10	3.10	1.54	2.36	0.78	45.57	2.39	0.93	0.99	

Visualisation par plis pour XGBoost

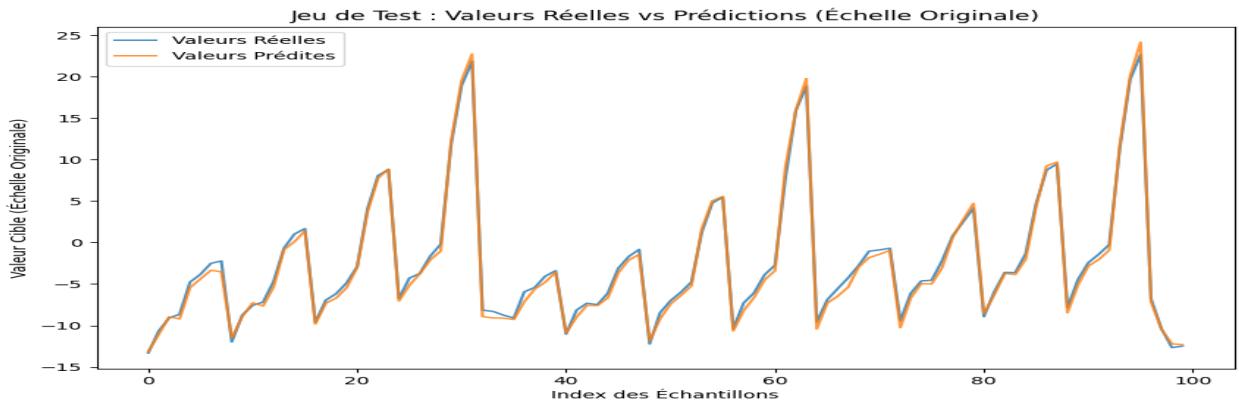
Pli5 :



Pli7 :



Pli10 :



Les graphes montrent que XGBoost est performant dans la prédiction des valeurs cibles pour les séries temporelles. Le modèle est particulièrement efficace pour les plis Pli 5 et Pli 7, avec une bonne généralisation et un faible écart entre les valeurs réelles et prédictives.

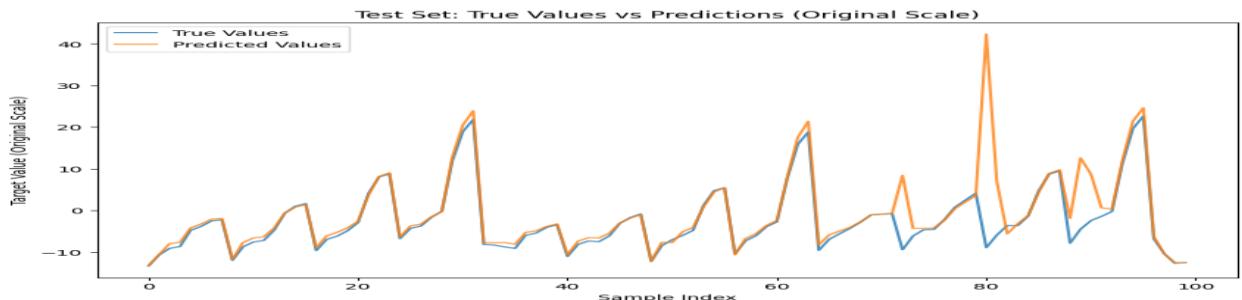
### Présentation du tableau des résultats obtenus dans les différents plis et modèles deep learning

#### LSTM

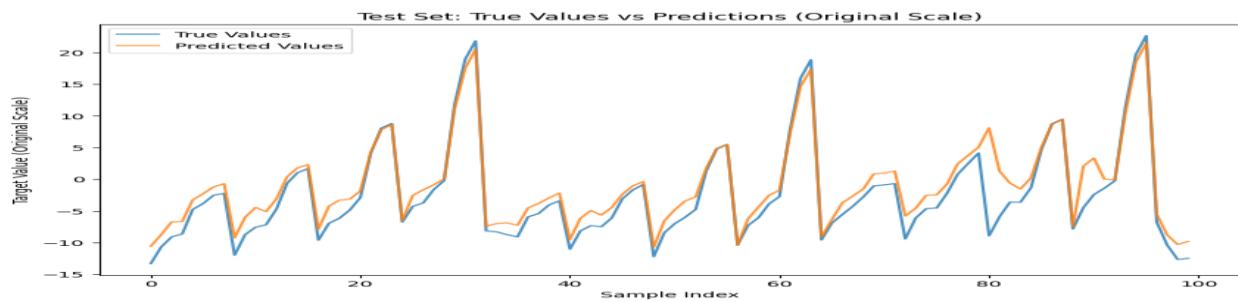
Méthode	MSE		MAE		RMSE		R CARRÉ		Temps	Performance
Train/Test	Train	Test	Train	Test	Train	Test	Train	Test		
Pli5	64.27	7.88	4.83	1.56	6.25	2.80	0.92	0.98	7min58s	
Pli7	120	6.3	6.34	1.38	8.91	2.51	0.58	0.98	9min56s	Meilleure
Pli10	552	12	11.74	2.21	13.96	3.51	0.16	0.97	8min23s	

#### Visualisation des plis :

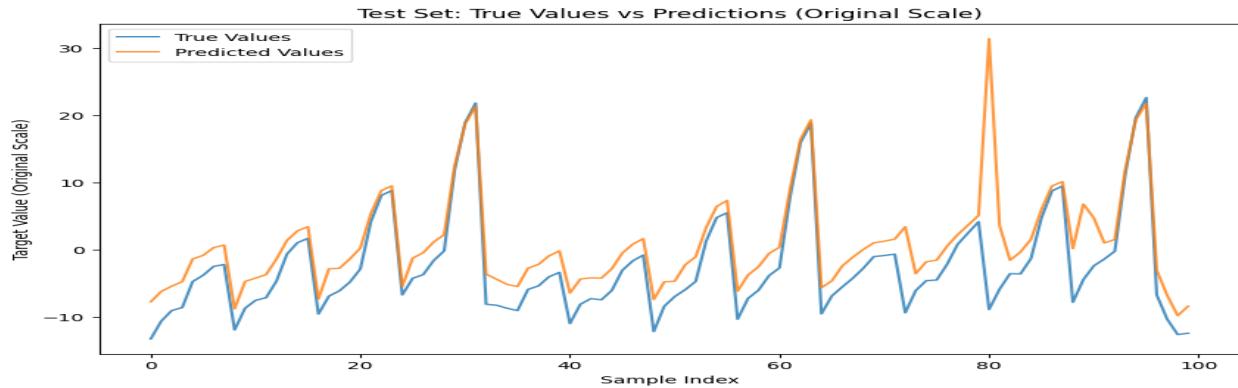
##### Pli5



##### Pli7



Pli10



Pli7 est le meilleur modèle dans ce contexte. Il offre un excellent compromis entre précision, robustesse, et généralisation sur les données de test.

Pli5 est performant mais moins précis dans les extrêmes.

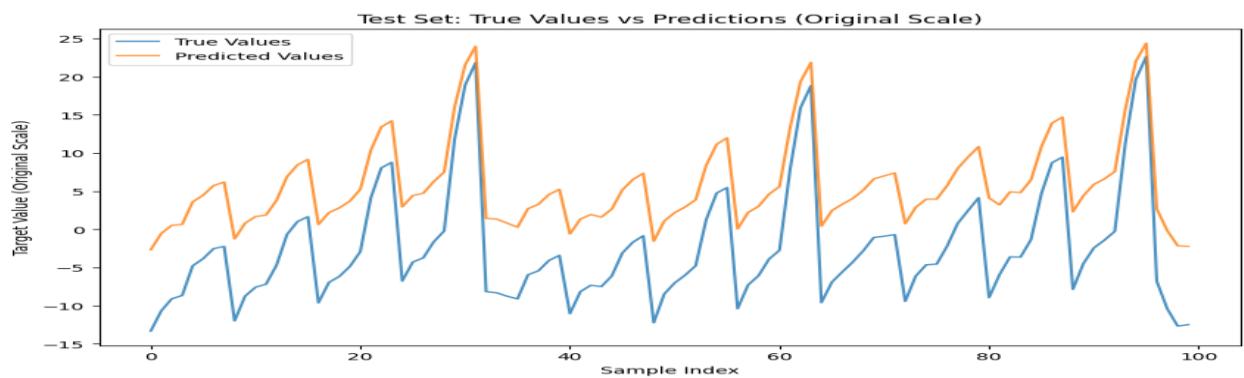
Pli10 souffre probablement de sur-apprentissage, avec une perte de précision sur les données de test.

### Bi-LSTM

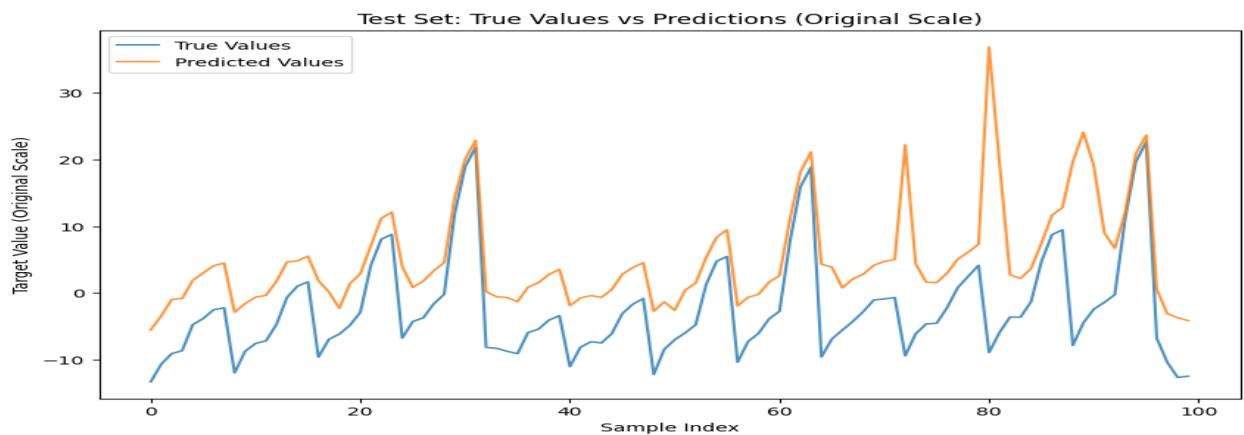
Méthode	MSE		MAE		RMSE		R CARRÉ		Temps	performance
Train/Test	Train	Test	Train	Test	Train	Test	Train	Test		
Pli5	80.53	13.06	6.30	2.54	7.66	3.61	0.91	0.97	14min8s	Meilleure
Pli7	155	31.21	7.47	4.48	9.94	5.58	0.62	0.93	10min23	
Pli10	379	23.21	11.14	3.19	14.51	4.81	0.18	0.94	10min26s	

Visualisation par plis :

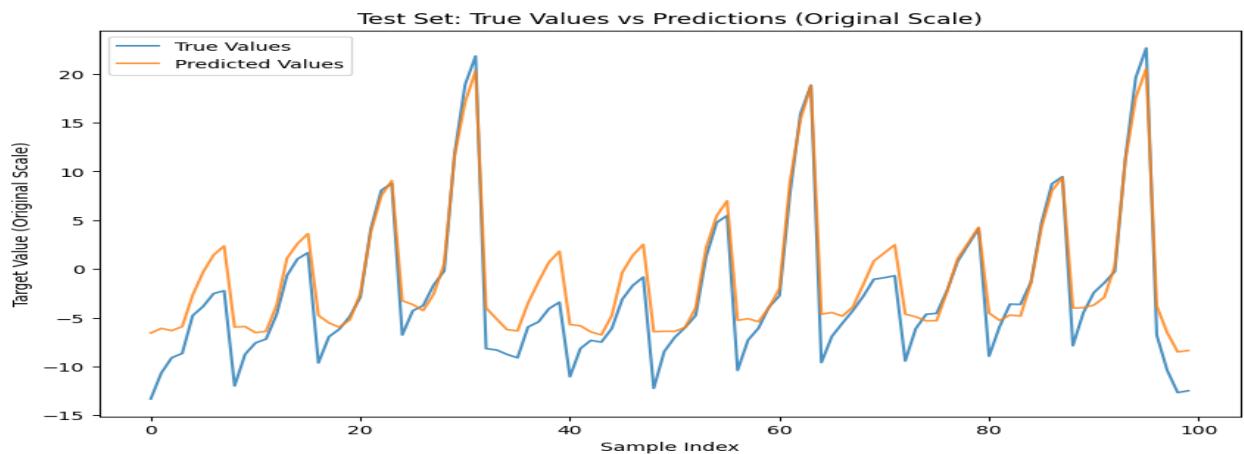
5plis



**7plis**



**10plis**



Pli5 est la configuration la plus performante pour le Bi-LSTM, offrant une précision supérieure et une excellente généralisation sur les données de test.

Pli7 montre des performances acceptables, mais son efficacité est inférieure à celle de Pli5.

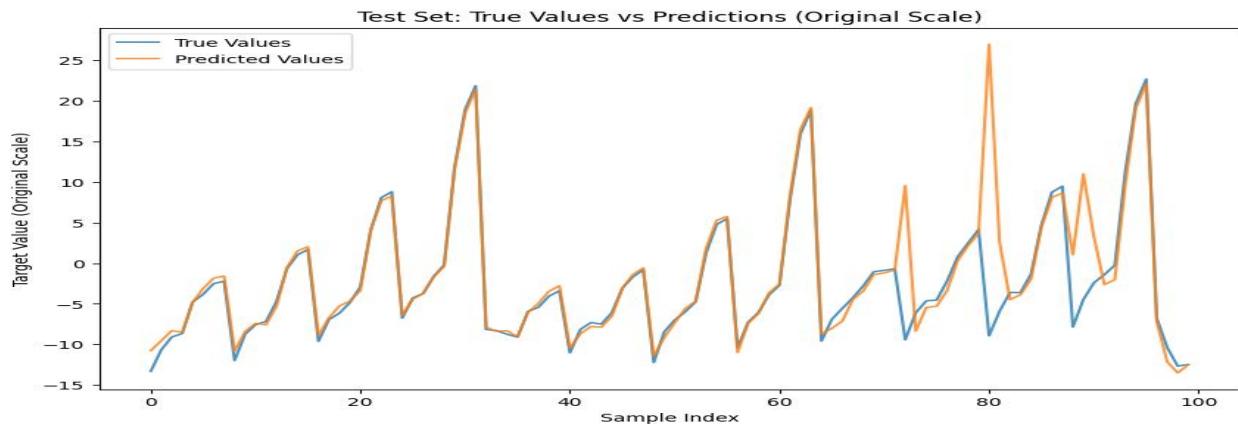
Pli10 a des performances respectables mais reste limité par rapport à Pli5.

## GRU

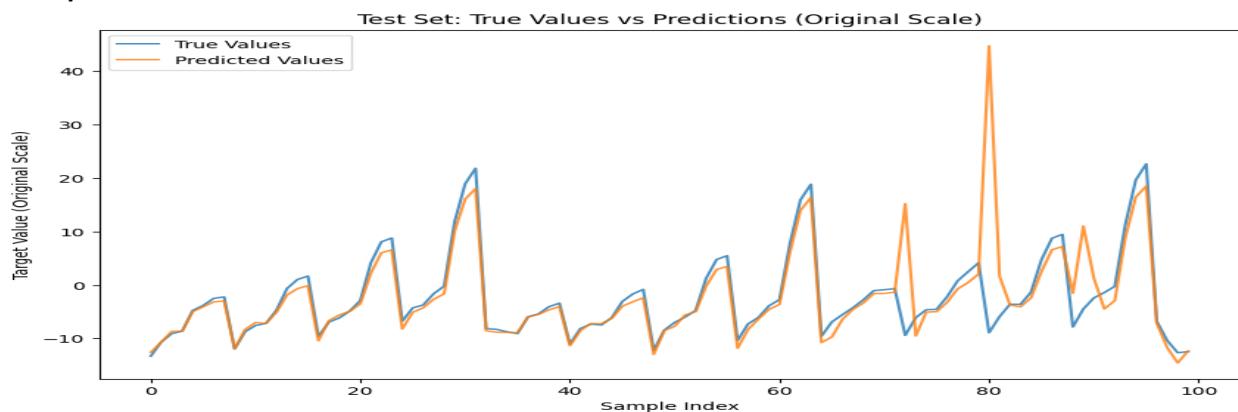
Méthode	MSE		MAE		RMSE		R CARRÉ		Temps	performance
train/test	train	test	train	test	train	test	train	test		
Pli5	26.66	8.25	3.04	2.19	1.091	2.87	0.96	0.98	9min38s	Meilleure
Pli7	57.57	20.56	4.56	3.72	6.16	4.53	0.83	0.95	11min47s	
Pli10	320	18.78	10.34	2.43	12.51	4.33	0.50	0.95	16min51s	

Visualisation par plis :

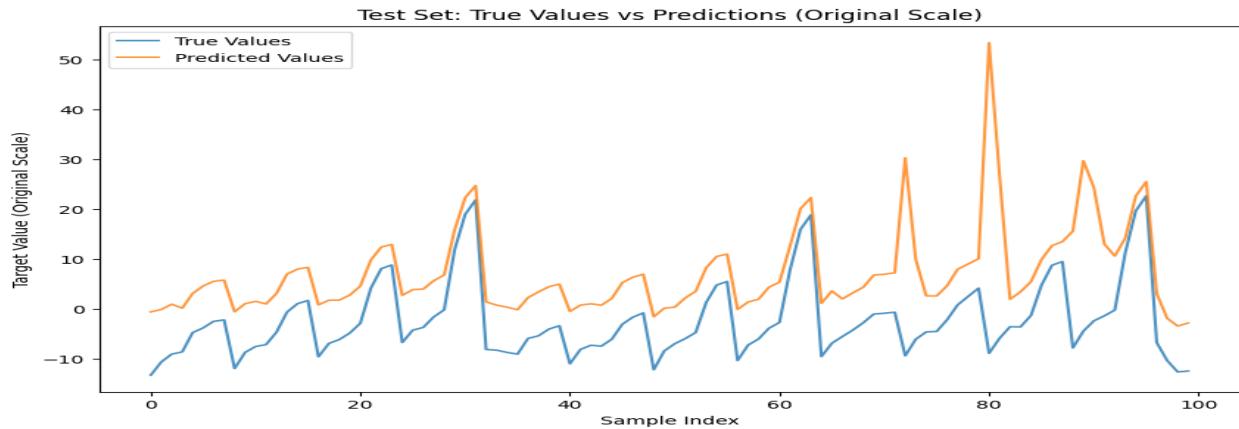
Par 5 plis



Par 7 plis



Par 10 plis



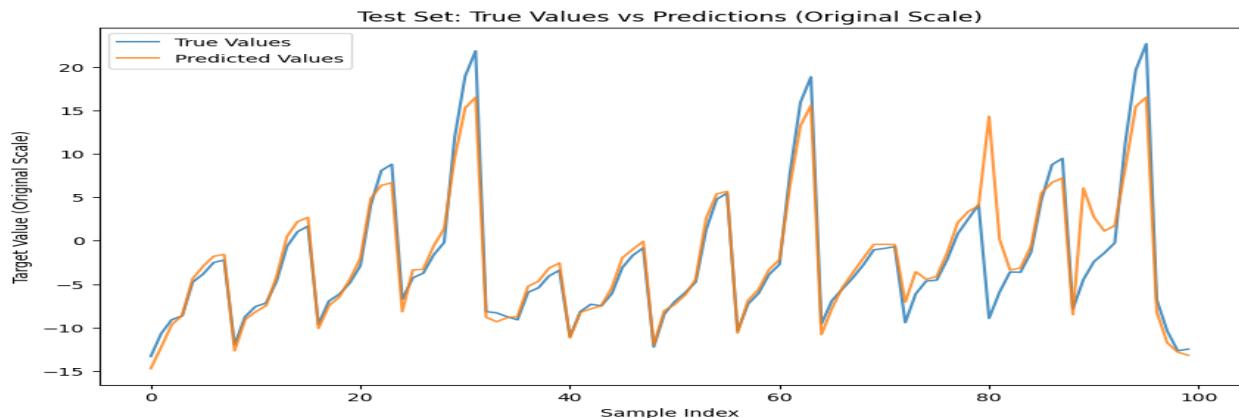
Le modèle GRU montre globalement de bonnes performances sur les séries temporelles, avec des prédictions précises et des métriques satisfaisantes pour le Pli 5 et le Pli7 offre des résultats acceptables mais est moins performant que Pli5, surtout sur les prédictions fines, tandis que des ajustements mineurs pourraient renforcer la stabilité du Pli 10.

## RNN

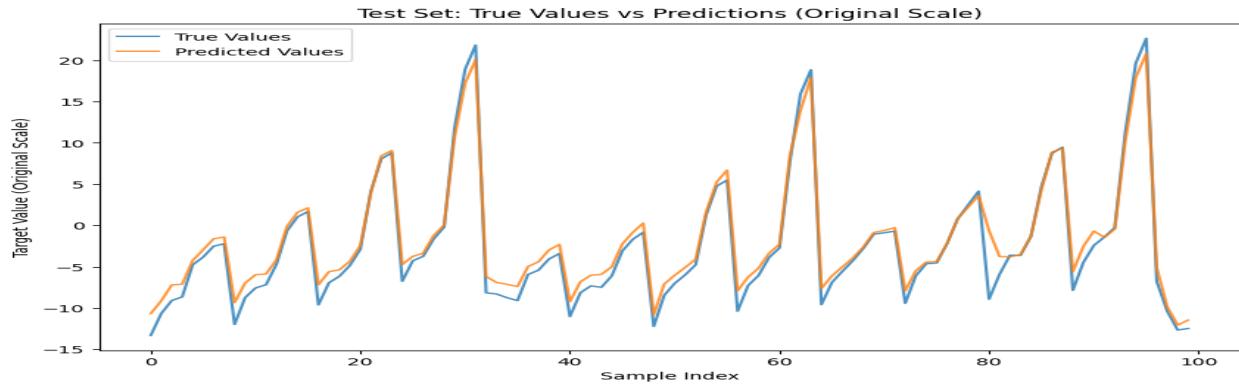
Méthode	MSE		MAE		RMSE		R CARRÉ		Temps	performance
train/test	train	test	train	test	train	test	train	test		
Pli5	37.09	56.60	4.59	6.37	6.00	7.52	0.94	0.87	6min48s	
Pli7	39.22	6.75	3.59	2.00	4.32	2.59	0.90	0.98	9min1s	
Pli10	130	7.10	5.60	1.66	6.74	2.66	0.75	0.98	13min56s	

## Visualisation par plis :

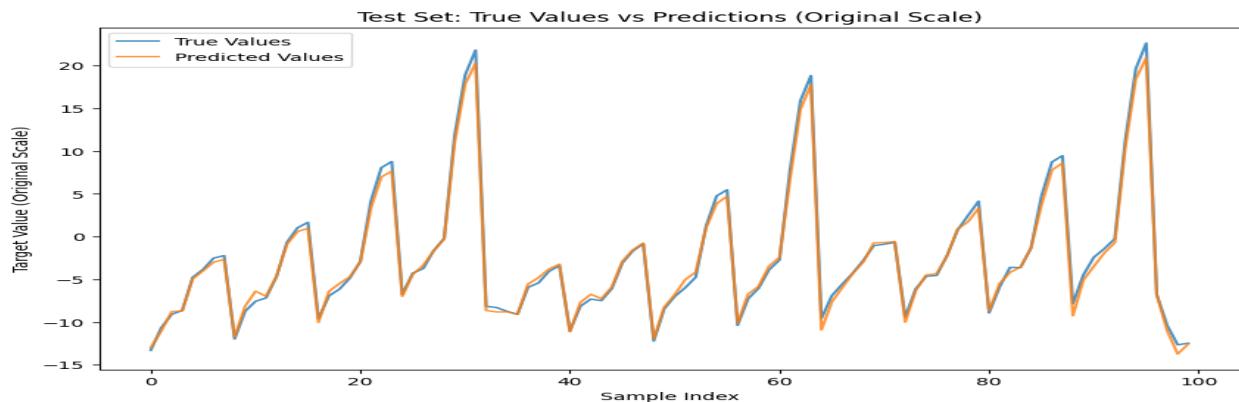
### Par 5plis



### Par 7 plis



### Par 10 plis



Pli7 est la configuration la plus performante pour le RNN. Il offre un excellent équilibre entre précision et temps d'entraînement raisonnable.

Pli10 est une alternative solide, mais son temps d'entraînement plus long peut être un inconvénient.

Pli5, malgré son temps d'entraînement rapide, n'est pas recommandé en raison de ses performances inférieures sur le test.

### Conclusion :

A	B	C	D	E	F	G	H	I
Type de Réseau	Configuration	MSE (test)	MAE (test)	RMSE (test)	R2(test)	Temps d'Entraînement	Justification	
LSTM	Pli7	6.3	1.38	1.38	0.98	9min56s	Meilleur compromis entre précision et généralisation.	
Bi-LSTM	Pli5	13.06	2.54	3.61	0.97	14min8s	Très précis avec un R2 élevé et des erreurs minimales.	
GRU	Pli5	8.25	2.19	2.87	0.98	9min38s	Faibles erreurs et excellent R2 avec un temps rapide.	
RNN	Pli7	6.75	2.0	2.59	0.98	9min1s	Meilleur équilibre entre précision et robustesse.	
Xgboost	pli7	1.56	0.53	1.25	0.99	9s		

Le modèle XGBOOST avec le pli 7 offre d'excellents résultats.

Et pour les modèles de deep learning :

Parmi les configurations, LSTM avec Pli7 et GRU avec Pli5 semblent les plus performants pour leurs faibles erreurs et un très bon R<sup>2</sup> sur les données de test, avec des temps d'entraînement raisonnables.

## Déploiement

Vous disposez d'un code pour faire fonctionner le modèle :

Vous aurez à rentrer les informations des données de temps réel et les modèles vous feront la prediction.

### Multi-Model Prediction

Entrez les valeurs des features pour obtenir les prédictions IRR avec plusieurs modèles.

LAT	<input type="text" value="0"/>
LON	<input type="text" value="0"/>
YEAR	<input type="text" value="0"/>

{...} Results



On a tenté de le déployé de façon permanente sur Huggins spaces. Et pour des raisons que j'ignore,

Les résultats ne s'affichent pas

Vous pouvez avoir accès sur ce lien : [IrrigationPrediction - a Hugging Face Space by billotanou](#)

## References

[Fouta-Djalon — Wikipédia \(wikipedia.org\)](#)

[Programme de développement local et de réhabilitation agricole au Fouta-Djalon \(PRAADEL\) \(ifad.org\)](#)

[b7c23d4c-bf5c-0218-955f-7bf9da974885 \(ifad.org\)](#)

<https://www.fao.org/>

[Calcul ETP \(Penman-Monteith\) :: Centre d'aide de l'application SICLIMA \(inrae.fr\)](#)

[Une méthode simple basée sur la température maximale \(ipcinfo.org\)](#)

[PDF/TABLES/Annexe1fra.pdf \(firebasestorage.googleapis.com\)](#)

[photo Moyenne guinee carte - Recherche Images \(bing.com\)](#)

[Banana | Land & Water | Food and Agriculture Organization of the United Nations | Land & Water | Food and Agriculture Organization of the United Nations \(fao.org\)](#)

[merra — merra documentation \(merra2.readthedocs.io\)](#)

[Time series / date functionality — pandas 1.4.0 documentation](#)

[Mise en œuvre de la prévision de séries temporelles pour les prévision – peerdh.com](#)

<https://colab.research.google.com/drive/1xnYLHozOmKwz1QNbbQiy3eTmR0WuQBC-?usp=sharing>

[Forum\\_ProposalTemplate\\_2023.pdf](#)