



DataScientest • com

*Rapport Technique d'évaluation*

# Kapy

Promotion: Data Scientist - Novembre 2022

Participants : Arnaud Machefel, Olivier Lauffenburger, Pierre Le Bert,  
Marion Kaisserlian

# Table des matières

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Contextualisation.....</b>	<b>3</b>
<b>3. Jeu de données.....</b>	<b>4</b>
a. Source des données.....	4
b. Exploration des données.....	5
i. Vue générale.....	5
ii. Observations principales.....	6
c. Traitement et augmentation des données.....	12
i. Avec des données externes.....	12
ii. En remplaçant les NaN.....	13
iii. Avec quelques données calculées.....	13
iv. Suppression de variables.....	14
<b>4. Modélisations.....</b>	<b>15</b>
a. Classification binaire sans rééchantillonage.....	15
i. Normalisation des données.....	15
ii. Modèles naïfs.....	15
1. Modèle "Désertique" qui prédit qu'il ne pleut jamais:.....	15
2. Modèle "Parfait" qui prédit la pluie.....	16
iii. Etude du "KNeighborsClassifier".....	16
1. Simulations en faisant varier les principaux paramètres.....	19
2. Analyse des résultats obtenus avec cette batterie de simulations.....	23
3. Proposition d'amélioration: Création d'une 'metric' personnalisée.....	25
4. Proposition d'amélioration: Création d'une fonction de pondération.....	28
5. Proposition d'amélioration: Modifier le jeu de données en ajoutant une pondération des variables via l'information mutuelle.....	31
iv. Utilisation d'une PCA sur KNN.....	33
1. Simulation PCA sur 12 composantes principales.....	34
2. Simulation PCA sur les 28 composantes principales (sans pondération).....	36
3. Simulation PCA sur les 28 composantes principales (avec pondération).....	37
4. Conclusion sur l'utilisation de la PCA avec KNeighborsClassifier.....	37
v. Utilisation de KNeighborsClassifier sur un KMeans.....	38
vi. Conclusions sur l'utilisation du KNeighborsClassifier.....	40
b. Classification binaire avec rééchantillonage.....	41
c. Classification multi-classes.....	42
d. Régression.....	42
i. Echelle quotidienne.....	42
ii. Echelle mensuelle.....	45
iii. Combinaison de modèles.....	47
iv. Interprétabilité.....	49
e. Séries temporelles.....	50
i. Introduction.....	50
ii. Préparation des données.....	51
iii. Estimation des paramètres ARIMA.....	51

iv. Modèle SARIMAX.....	52
Estimation des paramètres SARIMA.....	52
Résultats.....	53
v. Modèle ARIMA.....	54
Résultats sur la ville d'Albury.....	55
vi. Ajustement du seuil.....	56
Résultats sur la ville d'Albury.....	56
Visualisation de l'ajustement du seuil.....	57
vii. Calcul sur toutes les villes.....	58
viii. Détermination de la pluie pour le dernier jour.....	59
Résultats sur l'ensemble des villes.....	59
ix. Interprétation du modèle.....	59
f. Réseaux de neurones récurrents.....	61
i. Interprétabilité du modèle RNN.....	62
Interprétation globale avec Skater.....	62
Calcul de dépendance partielle avec Skater.....	64
Interprétation locale avec LIME.....	64
<b>5. Bilan et suite du projet.....</b>	<b>66</b>
<b>Annexes - Notebooks.....</b>	<b>68</b>

# 1. Introduction

Dans le cadre de la formation DataScientest - Data Scientist, nous formons une équipe de 4 apprentis venant d'horizons différents mais avec la même envie de découvrir la data science.

Notre objectif commun sera de prédire les risques de précipitations en Australie en utilisant des techniques de machine learning.

Même si nous n'avons pas d'expertise particulière en météorologie, nous avons choisi ce sujet chacun pour des raisons autant liées à nos métiers qu'à notre curiosité:

Marion	Pas d'intérêt métier mais curiosité sur le sujet météo
Olivier	Intérêt métier en particuliers sur les méthodes d'analyse et de prédiction des séries temporelles
Pierre	Pas d'utilisation possible directe des modélisations du projet KAPY. Par contre, les méthodes d'analyse de données étudiées et les "classifiers" utilisés pourront être repris dans un cadre professionnel (quelques applications au CRM dans le cadre assurantiel : "next best action", rebond commercial, actuariat, marketing et ciblage pour la prospection, prévenir les résiliations par des actions de rétention ciblées)
Arnaud	Pas d'intérêt métier mais curieux de m'essayer au machine learning sur des données météos

Afin de faire avancer le projet au mieux, nous nous sommes organisés différemment selon les phases du projet:

- Exploration des données et feature engineering: travail en parallèle puis mise en commun nos résultats
- Modélisation + interprétabilité: répartition des différents types de modélisations entre chacun d'entre nous:
  - Classification binaire: Pierre
  - Classification binaire avec ré-échantillonnage: Marion
  - Classification multi-classe: Olivier
  - Régression: Arnaud
  - Série temporelle: Olivier
  - Réseaux de neurones: Olivier
- Conclusion: travail collectif

# 2. Contextualisation

GrouseLager est un brasseur qui fabrique et distribue sur le territoire Australien. Les départements de production et d'achats souhaitent intégrer les prédictions météorologiques afin d'optimiser les niveaux de production, besoins en matières premières et stockages.

L'entreprise a besoin de réduire ses coûts. Pour celà, elle identifie en particulier le besoin de mieux prévoir les périodes de pluies/sécheresse afin d'éviter des coûts de production inutiles (irrigation, arrosage...) et de pallier efficacement à des périodes non-pluvieuses qui pourraient impacter les récoltes négativement.

Cependant, la prédiction météorologique est particulièrement compliquée:

"Les pluies en Australie posent un problème encore plus délicat que leur faible valeur moyenne : elles sont imprévisibles. Dans de nombreuses régions agricoles du monde, la saison à laquelle la pluie tombe est prévisible d'année en année [...] Sur la plus grande partie du territoire australien, au contraire, les pluies dépendent de ce qu'on appelle l'ENSO (El Niño Southern Oscillation), à savoir le fait que la pluie n'est pas prévisible d'année en année sur une décennie et est encore plus imprévisible de décennie en décennie." (Effondrement - Jared Diamond)

"L'analyse à partir de données provenant d'une seule station météorologique est sans aucun doute la méthode d'analyse la plus ancienne et la plus répandue. Elle est utilisée par le marin, le berger, l'agriculteur et l'homme de la rue, avec des résultats qui font souvent l'envie du professionnel qui, dans certaines conditions, est susceptible d'être aveuglé par une masse de données météorologiques." (Principles of Meteorological Analysis, Chapitre 12 "Local Analysis" - Walter J. Saucier)

Nous sommes donc missionnés par GrouseLager pour trouver un moyen de les aider dans cette tâche. Nous nous appuierons sur un jeu de données contenant une dizaine d'années de relevé météorologique en Australie.

Le projet sera nommé 'KaPy' en référence à 'Kapi' qui signifie 'eau' pour les Aborigènes et Python qui est le langage de programmation utilisé pour nos modélisations.

Le présent rapport détaille:

- Le jeu de données
- Les modélisations réalisées et les résultats associés
- Les conclusions de l'étude

## 3.Jeu de données

### a. Source des données

Nos données sources sont issues du "Australian Government - Bureau of Meteorology" et ont été téléchargées sur Kaggle:

<https://www.kaggle.com/datasets/jspphyg/weather-dataset-rattle-package>

Par défaut, il faudra noter que ces données sont sous Copyright Commonwealth of Australia 2010 et ne sont donc pas disponibles librement pour une utilisation commerciale.

Nous avons utilisé quelques sources de données utilisable librement (Licence Creative Commons) afin d'augmenter les données:

- World Cities Database pour les latitudes et longitudes des villes:

<https://simplemaps.com/data/world-cities>

- Wikipedia pour les types de climat de chaque ville en Australie:

[https://en.wikipedia.org/wiki/Climate\\_of\\_Australia](https://en.wikipedia.org/wiki/Climate_of_Australia)

## b. Exploration des données

### i. Vue générale

Nous disposons d'un jeu de données conséquent avec un total de 145460 entrées; correspondant chacune à une observation quotidienne dans une des 59 stations d'observation météorologique australienne entre 2008 et 2017.

Pour chaque observation, nous disposons de 23 colonnes en incluant la variable cible:

#### Variables Catégorielles

1. **Date** [YYYY-MM-DD]: Date de l'observation au format YYYY-MM-DD
2. **Location** [string]: Lieu de l'observation
3. **RainToday** [Yes/No]: Les précipitations sont-elles > 1mm aujourd'hui ?
4. **WindGustDir** [string]: Direction de la plus forte rafale de vent dans les 24h jusqu'à 0:00 (en 16ème de direction N,S,E,O)
5. **WindDir9am** [string]: Direction du vent à 9:00 (en 16ème de direction N,S,E,O)
6. **WindDir3pm** [string]: Direction du vent à 15:00 (en 16ème de direction N,S,E,O)

#### Variables Quantitatives

7. **MinTemp** [°C]: Température minimale dans les 24h en °C
8. **MaxTemp** [°C]: Température maximale dans les 24h en °C
9. **Rainfall** [mm]: Précipitations dans les 24h en mm
10. **Evaporation** [mm]: Mesure standardisée d'évaporation dans la journée en mm
11. **Sunshine** [h]: Durée d'ensoleillement dans les 24h jusqu'à 9:00 en heure
12. **WindGustSpeed** [km/h]: Vitesse de la plus forte rafale de vent dans les 24h jusqu'à 0:00 en km/h
13. **WindSpeed9am** [km/h]: Vitesse du vent à 9:00 en km/h
14. **WindSpeed3pm** [km/h]: Vitesse du vent à 15:00 en km/h
15. **Humidity9am** [%]: Taux d'hygrométrie à 9:00
16. **Humidity3pm** [%]: Taux d'hygrométrie à 15:00
17. **Pressure9am** [hPa]: Pression atmosphérique à 9:00 en hPa
18. **Pressure3pm** [hPa]: Pression atmosphérique à 15:00 en hPa
19. **Cloud9am** [octa]: Fraction du ciel ennuagé à 9:00 (entier de 0 à 8 + la valeur spéciale 9 si le ciel ne peut être observé)
20. **Cloud3pm** [octa]: Fraction du ciel ennuagé à 15:00 (entier de 0 à 8 + la valeur spéciale 9 si le ciel ne peut être observé)
21. **Temp9am** [°C]: Température à 9:00 en °C
22. **Temp3pm** [°C]: Température à 15:00 en °C

Variable cible

23. **RainTomorrow** [Yes/No]: Les précipitations seront-elles > 1mm demain ?

## ii. Observations principales

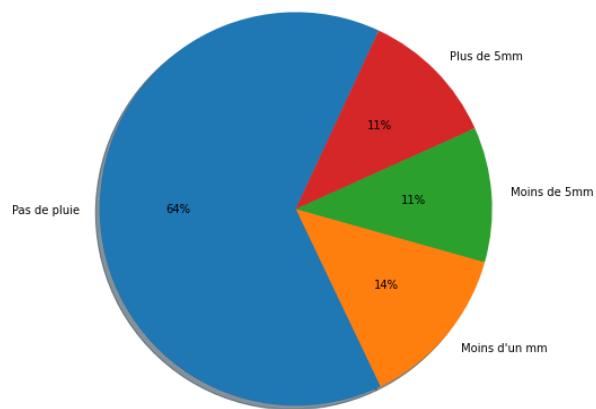
### Type de problème:

A première vue, il s'agit d'un problème de classification avec la variable binaire RainTomorrow. On pourra cependant utiliser la variable RainToday afin de transformer le problème en un problème de régression.

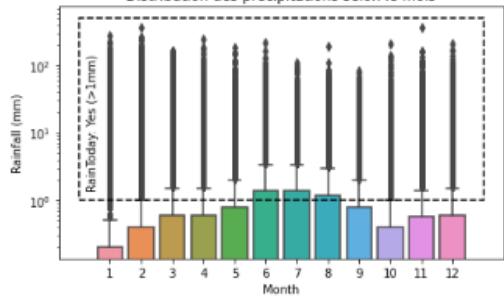
### Répartition de la variable cible:

- Le jeu est déséquilibré avec seulement environ 20% de jours de pluie.
- La plupart des précipitations sont assez faible (<10mm) mais nous observons aussi des valeurs extrêmes très élevées (jusqu'à >300mm)
- Il y a une saisonnalité dans la quantité de précipitation tout au long de l'année

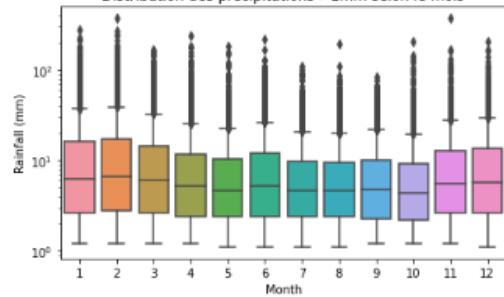
Distribution des précipitations



Distribution des précipitations selon le mois



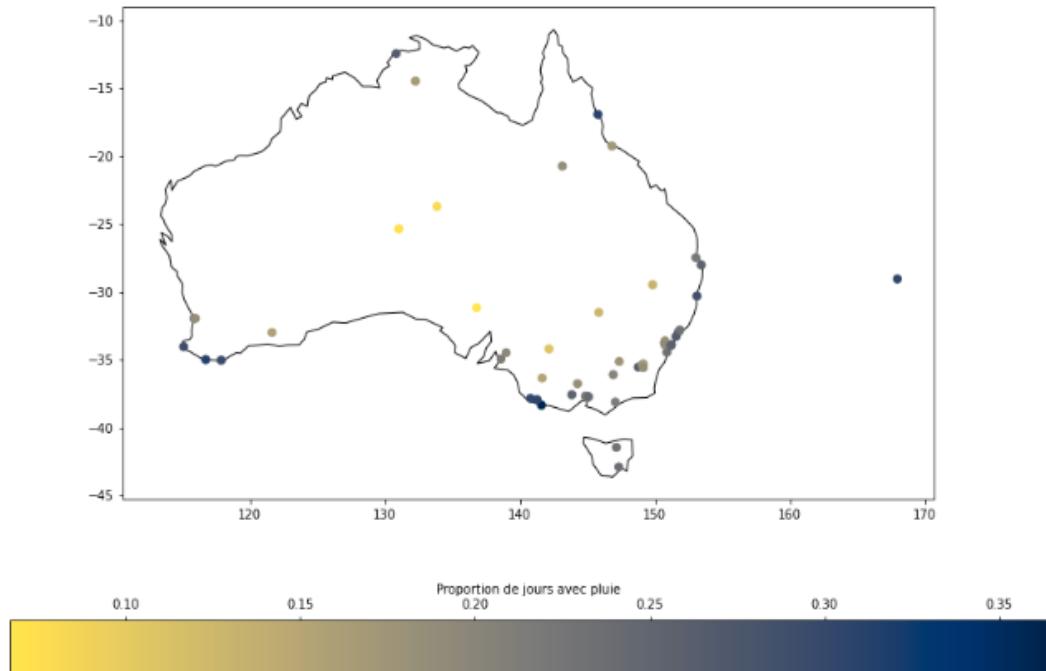
Distribution des précipitations >1mm selon le mois



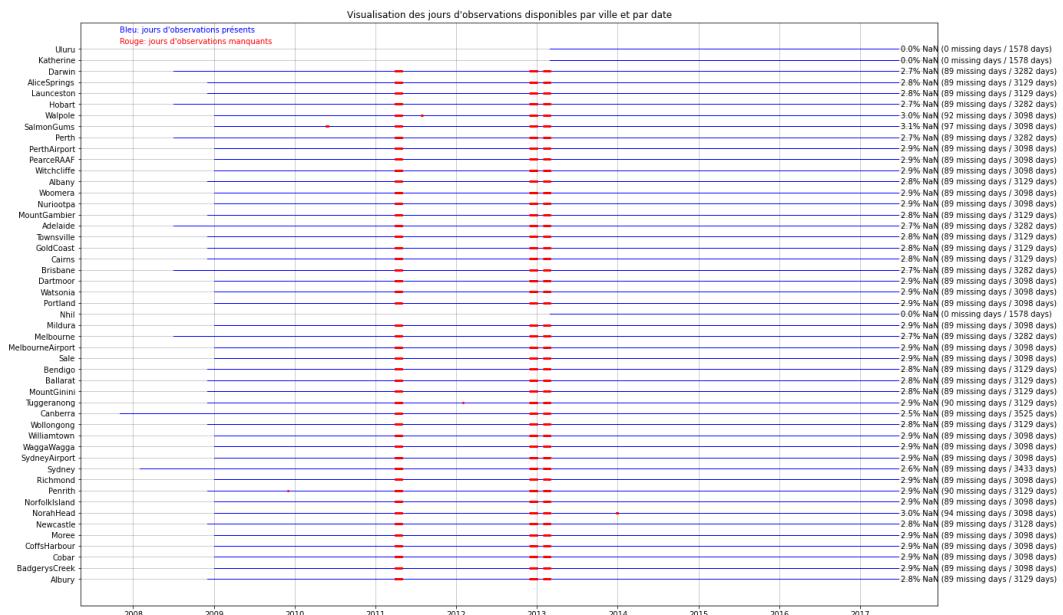
## Lieux d'observations:

Les différents lieux d'observation couvrent toutes la surface de l'Australie; qui compte plusieurs climat très différents (voir lien wikipedia en 3.a).

La majorité des lieux d'observation est concentrée sur la région sud-est. Certaines stations sont très isolées.

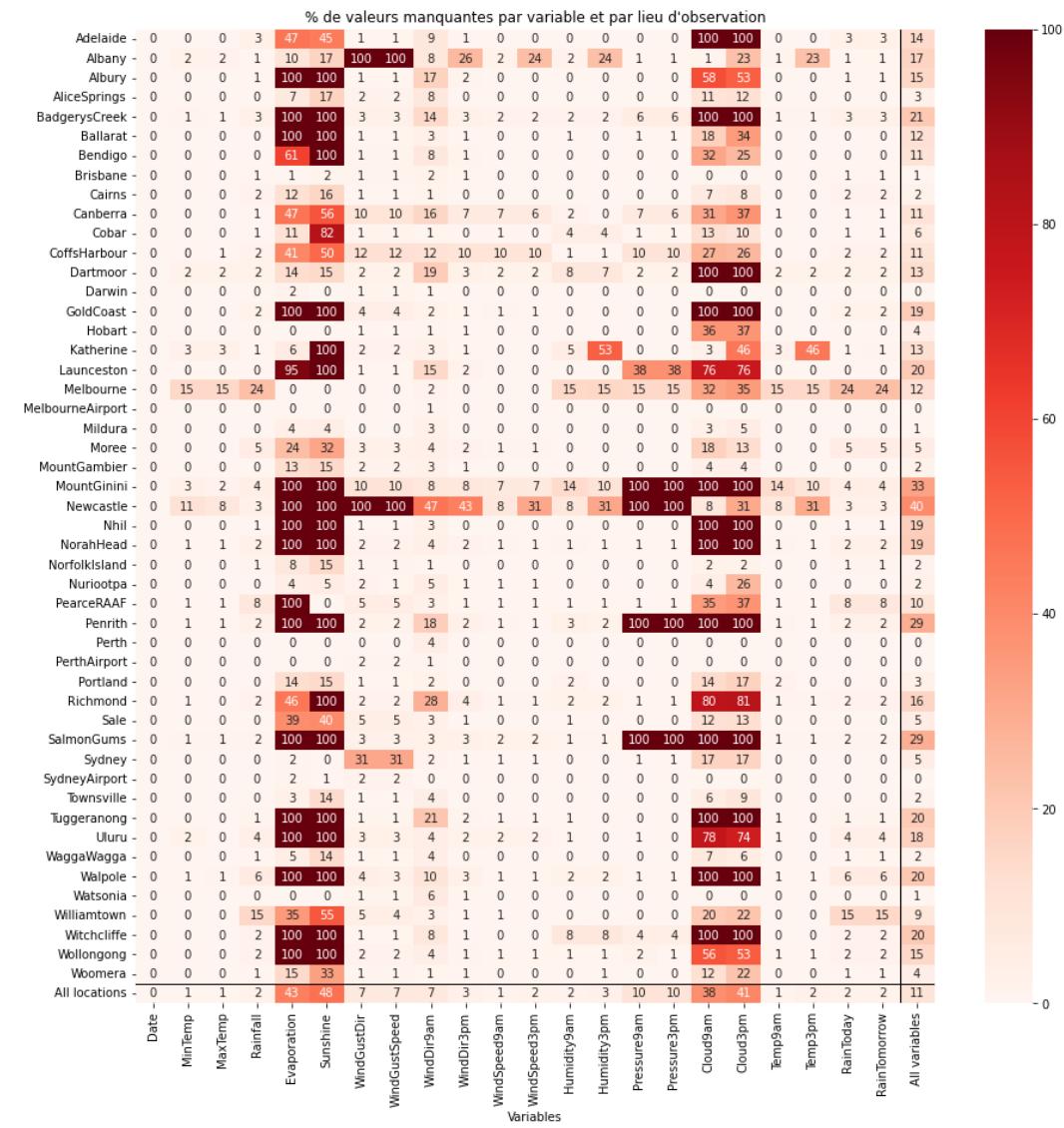


Les lieux d'observations n'ont pas tous le même historique de relevé:



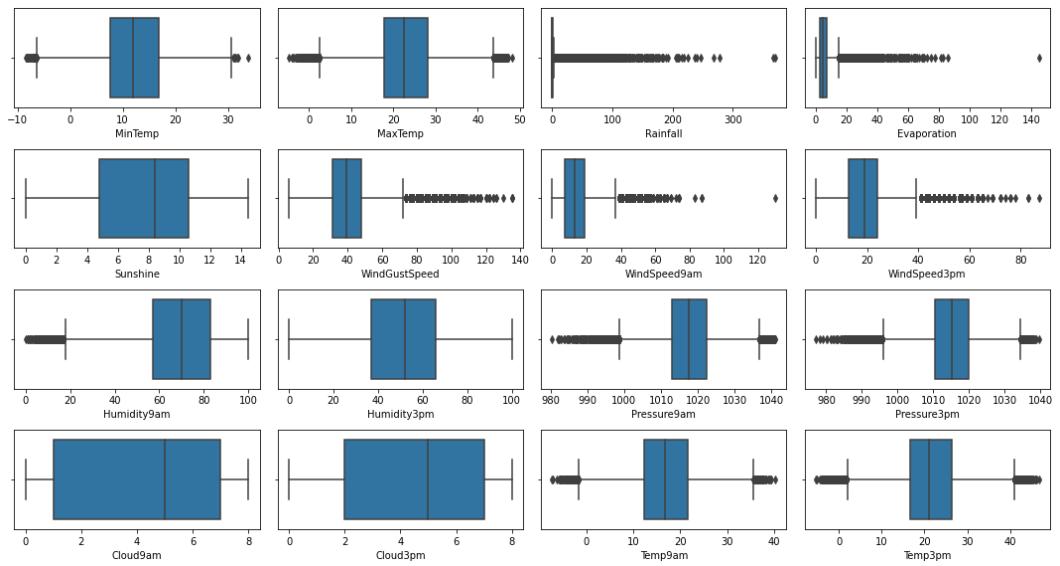
## Valeurs manquantes

Nous observons beaucoup de valeurs manquantes (NaN):



## Valeurs extrêmes:

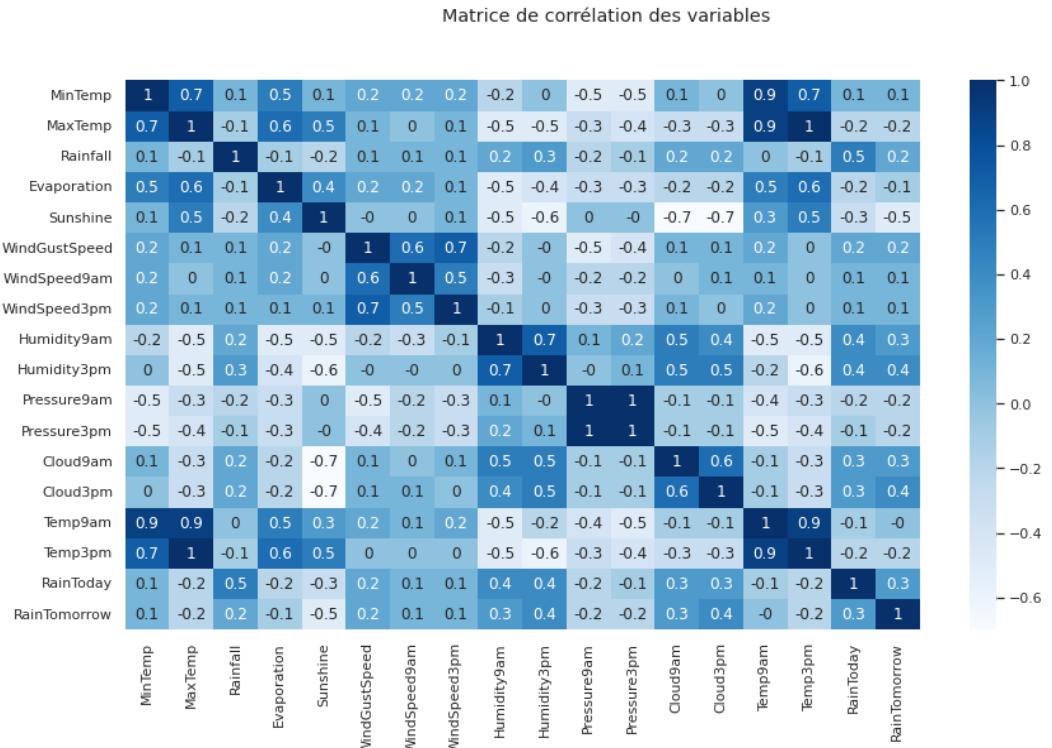
Nous n'observons pas de valeurs aberrantes. Cependant, nous avons des valeurs extrêmes sur quelques variables: vitesse du vent, évaporation et précipitation.



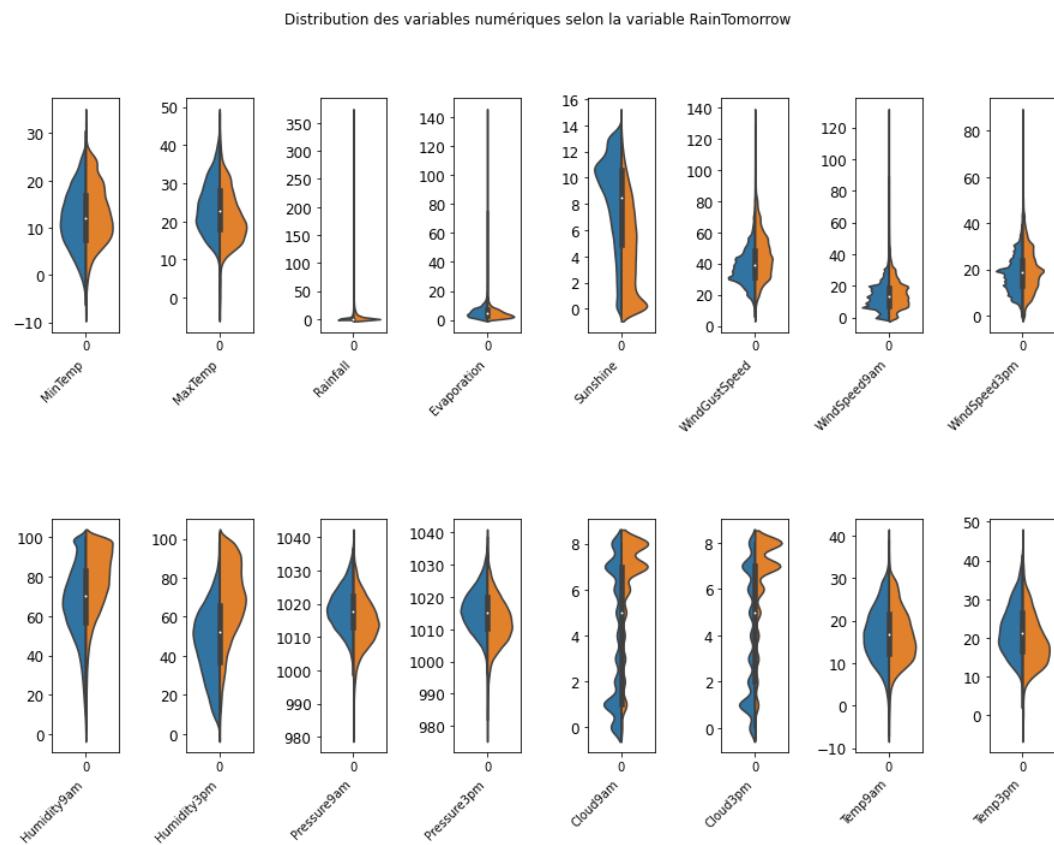
## Corrélation des données à la variable cible:

Cette matrice de corrélation nous permet tout d'abord de noter qu'aucune de nos variables n'a de corrélation linéaire importante ( $>0.5$ ) avec la variable cible.

Certaines valeurs (9am, 3pm) semblent parfois assez corrélées entre elles.

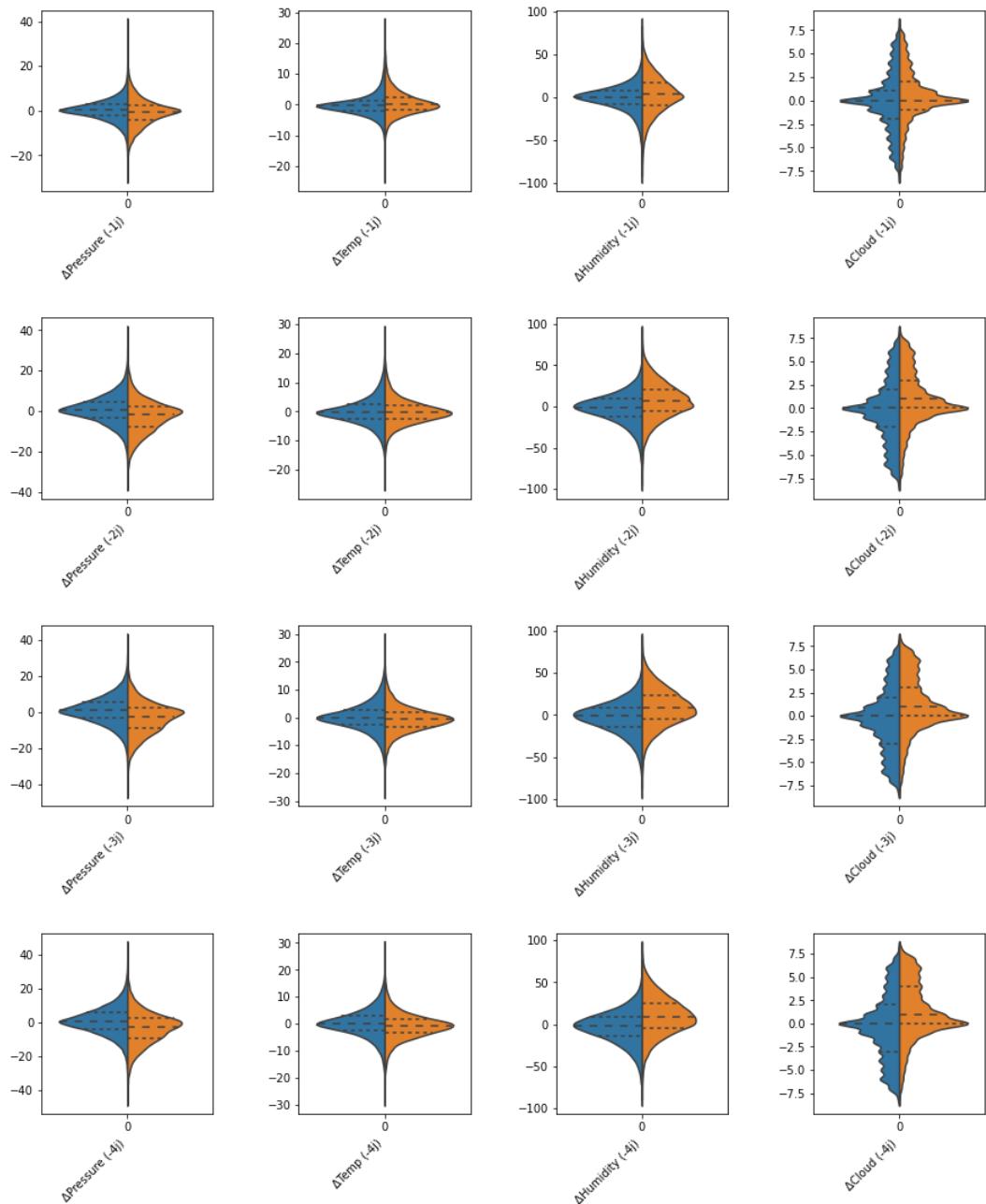


Une analyse graphique semble indiquer que quelques variables sont belles et bien corrélées à la variable cible:

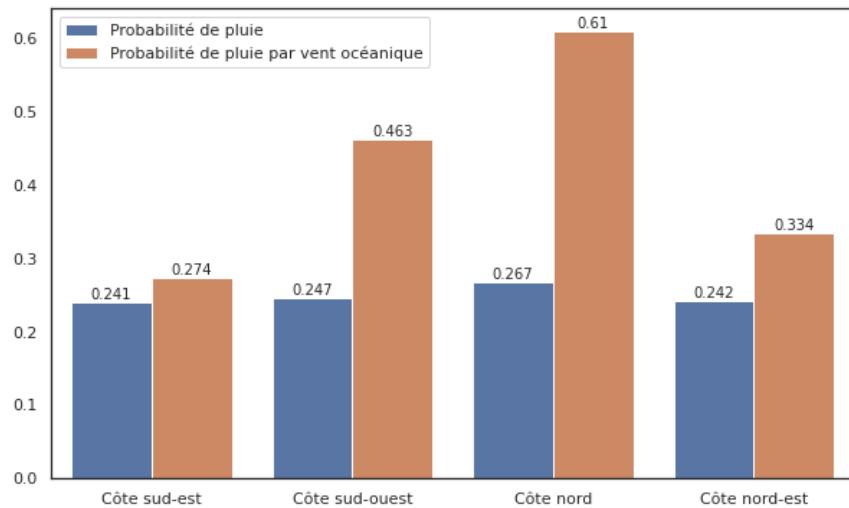


On observe que la différences de certaines valeurs physiques (pression, température, humidité) sur quelques jours semblent indiquer l'apparition de précipitations:

Distribution de la variation dans le temps des variables numériques selon la variable cible



La direction du vent semble favoriser (ou non) l'apparition de pluie:



## c. Traitement et augmentation des données

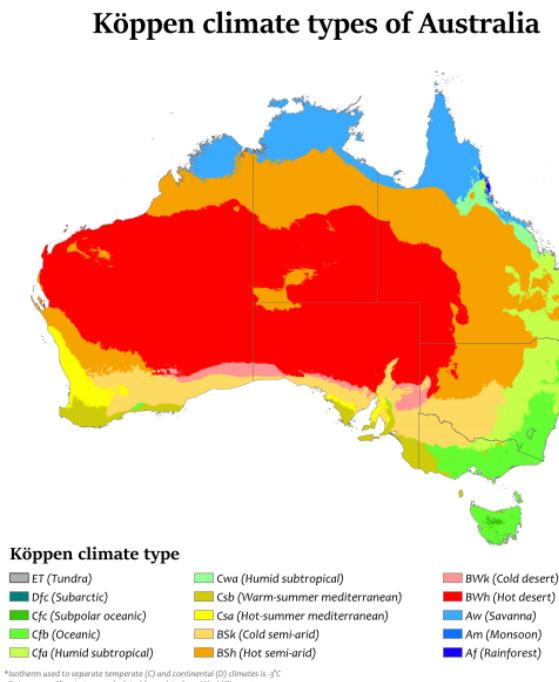
Suite aux observations réalisées dans l'exploration des données, nous avons jugé opportun d'augmenter notre jeu de données:

- Avec des données externes
- En remplissant des NaN
- Avec des données calculées
- En supprimant des colonnes inutiles/redondantes

### i. Avec des données externes

**Latitude et Longitude:** Données récupérées de "World Cities Database"

**Données de climat :** Données récupérées de Wikipedia (voir 3.a)



## ii. En remplissant les NaN

### Pressure:

Etant donné que Pressure3pm et Pressure9am sont assez corrélés, nous remplissons les NaN de Pressure3pm avec Pressure9am .

### Humidity:

Etant donné que Humidity3pm et Humidity9am sont assez corrélés, nous remplissons les NaN de Humidity3pm avec Humidity9am.

### Temp9am:

Etant donné que Temp9am et MinTemp sont assez corrélés, nous remplissons les NaN de Temp9am avec MinTemp.

## iii. Avec quelques données calculées

En réalisant différentes observations sur nos variables de base, nous sommes capables de sélectionner quelques variables calculées qui ne sont pas corrélées linéairement avec les variables de base et qui semblent avoir un impact sur la variable cible:

Variable calculée	Calcul
diffTempMinMax	= MaxTemp - MinTemp
diffWind3pm9am	= WindSpeed3pm - WindSpeed9am
diffPressure9am3pm	= Pressure3pm - Pressure9am
diffHumidity9am3pm	= Humidity3pm - Humidity9am
DeltaP_1d, DeltaP_2d, DeltaP_3d	= Pressure3pm[today] - Pressure3pm[today-(1,2,3)day]
DeltaH_1d, DeltaH_2d, DeltaH_3d	= Humidity3pm[today] - Humidity3pm[today-(1,2,3)day]
WindDirInfluence	= Coefficient représentant l'influence de la direction du vent sur la probabilité d'avoir de la pluie (cf Notebook pour les calculs)
consecutiveRainingDays	= Nombre de jour consécutif de pluie (cf Notebook pour les calculs)

#### iv. Suppression de variables

Variable supprimées	Explication
'diffTemp3pm9am', 'Temp3pm', 'MinTemp', 'MaxTemp'	diffTempMinMax et Temp9am sont suffisantes
'diffWindGust9am', 'diffWindGust3pm', 'WindSpeed9am', 'WindSpeed3pm'	diffWind3pm9am et WindGustSpeed sont suffisantes
'WindGustDir', 'WindDir3pm', 'WindDir9am'	L'information concernant la direction des vents et l'impact sur la pluie est intégré dans WindDirInfluence
'Pressure9am', 'Pressure3pm'	Remplacé par Pressure
'Humidity9am', 'Humidity3pm'	Remplacé par Humidity
'RainToday'	RainFall est suffisant
'Evaporation', 'Sunshine', 'Cloud9am', 'Cloud3pm'	Quantité de NaN trop importante

# 4. Modélisations

Nous identifions plusieurs axes d'études pour la modélisation. Chacun de ces axes a été étudié en parallèle par chacun des membres de l'équipe. Les conclusions sont détaillées dans les sections ci-après.

Ces axes comprennent aussi bien des classifications (binaires et multi-classes) que des régressions.

## a. Classification binaire sans rééchantillonnage

### i. Normalisation des données

Nous allons utiliser un preprocessing de normalisation MinMax. La normalisation MinMax effectue le calcul suivant sur chaque variable :

$$X_{\text{std}} = (X - X.\min(\text{axis}=0)) / (X.\max(\text{axis}=0) - X.\min(\text{axis}=0))$$

$$X_{\text{scaled}} = X_{\text{std}} * (\text{max} - \text{min}) + \text{min}$$

Les valeurs des variables sont donc ramenées entre 0 et 1.

#### Pourquoi effectuer une normalisation ?

Prenons un exemple. Les plages de valeurs sont très grandes sur la pression et faibles sur la température. Aussi, sans renormalisation, la valeur de la pression prendra une part bien plus importante que la température dans les résultats de notre modélisation. Cette pondération est sans rapport direct avec l'importance réelle de la pression ou de la température sur la capacité à prédire la valeur de la "target".

Il est donc très important de normaliser nos variables.

### ii. Modèles naïfs

#### 1. Modèle "Désertique" qui prédit qu'il ne pleut jamais:

Un modèle qui prédirait qu'il ne pleut jamais aurait cette matrice de confusion :

		PREDICTIONS		Predicted	
		0	1	Negative	Positive
REALITE	0	18520	0	True Negative	False Positive
	1	5299	0	False Negative	True Positive

Et donc les éléments suivants :

- Accuracy =  $(TP+TN)/(TP+FP+FN+TN) ==> 0.777$
- Recall =  $TP/(TP+FN) ==> 0.0$
- Specificity =  $TN/(TN+FP) ==> 1.0$
- Le F1-Score et la Précision ne peuvent être calculés ici (division par 0).

Ce modèle "désertique" a déjà une accuracy de plus de 77% mais un rappel de 0.

L'Accuracy n'est donc pas le seul élément déterminant, car même pour le modèle "Désertique", on obtient déjà 77% de bonnes prédictions.

## 2. Modèle "Parfait" qui prédit la pluie

Ce modèle parfait aurait cette matrice de confusion :

		PREDICTIONS	
		0	1
REALITE	0	18520	0
	1	0	5299

Actual		Predicted	
		Negative	Positive
Negative	True Negative	False Positive	
	False Negative	True Positive	

Et donc les éléments suivants :

- Accuracy =  $(TP+TN)/(TP+FP+FN+TN) = 1.0$
- Precision =  $TP/(TP+FP) = 1.0$
- Recall =  $TP/(TP+FN) = 1.0$
- F1-Score =  $2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision}) = 1.0$
- Specificity =  $TN/(TN+FP) = 1.0$

**Notre objectif est donc de faire tendre vers 1 : l'Accuracy, la Precision, le Recall( rappel), le F1-Score et la Specificity.**

### iii. Etude du “KNeighborsClassifier”

Nous allons étudier ici le “classifier” : KNeighborsClassifier et son utilisation dans le cadre de Kapy.

#### Qu'est-ce que le KNeighborsClassifier ?

Il s'agit d'une méthode de classification des échantillons basée sur un calcul de la 'distance' entre un échantillon et ses plus proches voisins et puis par l'attribution de la classe sur la base du plus grand nombre de voisins ayant la même classe.

(<https://scikit-learn.org/stable/modules/neighbors.html#classification> )

#### KNeighborsClassifier sur RainTomorrow binaire

Nous commençons par tester un modèle simple sur KNeighborsClassifier afin de prédire la variable binaire RainTomorrow.

#### Détail des paramètres pour KNeighborsClassifier

Le classifier KNeighborsClassifier dispose de nombreux paramètres permettant d'optimiser les modélisations.

Nous allons passer en revue ces paramètres et la valeur attribuée pour cette première simulation. Par la suite nous tenterons une optimisation puis une étude détaillée de certains éléments du paramétrage. (ref :

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn-neighbors-kneighborsclassifier>

Paramètre	Remarques
<b><i>n_neighbors = 2</i></b>	Indique le nombre de voisins utilisés pour déterminer la classe de l'échantillon. Plus le nombre de voisins est grand et moins chaque voisin impacte le résultat. Par contre, ceci peut diminuer la facilité à classer les échantillons
<b><i>weights = "uniform"</i></b>	Indique le poids relatif de chaque voisin. Deux calculs sont proposés en standard :  "uniform" : chaque voisin a le même impact "distance" : l'impact de chaque voisin est inversement proportionnel à sa distance
<b><i>algorithm = "auto"</i></b>	Indique l'algorithme utilisé pour déterminer qui sont les k voisins. Quatre options sont possibles :  "brute" : utilise la 'force brute' [temps de calcul ~DxN]  "ball_tree": utilise l'algorithme BallTree [temps de calcul ~Dlog(N)]  "kd_tree": utilise l'algorithme KDTree [D<15 : temps de calcul ~Dlog(N); D>15 temps de calcul ~DxN]  "Auto" : voir plus bas.
<b><i>leaf_size = 30</i></b>	indique la taille des feuilles passées en paramètre à l'algorithme BallTree ou KDTree.  Impacte la durée des calculs. En effet, les algorithmes KD Tree et Ball Tree prévoient de passer à 'force brute' sur les n derniers échantillons composant les feuilles au bout de l'arbre (ici on repassera à brute force pour n= 30 (valeur conseillée)).  ref : <a href="https://github.com/scikit-learn/scikit-learn/blob/364c77e04/sklearn/neighbors/_classification.py#L195">https://github.com/scikit-learn/scikit-learn/blob/364c77e04/sklearn/neighbors/_classification.py#L195</a>
<b><i>p = 2</i></b>	Ce paramètre de 'puissance' n'est utilisé que pour la distance de 'Minkowski'
<b><i>metric = "minkowski"</i></b>	Indique la métrique utilisée pour calculer les distances entre chaque échantillon et n'importe lequel de ses voisins.

	Il existe diverses distances proposées par ce classifier en standard. (voir plus bas).
<b>metric_params = None</b>	Paramètres fournis lors de l'utilisation d'une fonction à la place des métriques disponibles.
<b>n_jobs = None</b>	Ressources machines allouées au calcul (-1 pour allouer toutes les ressources machines au calcul).

Remarque concernant le paramètre **algorithm = "auto"**:

ref : <https://scikit-learn.org/stable/modules/neighbors.html>

"Brute Force" est utilisé si n'importe laquelle de ces 5 conditions est vérifiée :

- Faible nombre d'échantillons N>>10000 (ko)
- La 'metric' utilisée n'est pas une metric !

En faisant appel à une fonction que nous pourrions créer:

=> Lors de notre premier essai, nous utilisons toutes les 'metrics' disponibles en 'standard', donc Brute Force sera utilisée par défaut. Lors d'un usage d'une metric créée par nos soins, nous pouvons forcer l'abandon de la 'force brute'. Ceci peut impacter négativement les temps de calcul si KD Tree est utilisé.

- D>15, Une réduction de dimensions en dessous de 15 (Via une PCA par exemple) pourra donc considérablement diminuer la durée des calculs.
- k>=N/2,
- La 'metric' utilisée n'est pas dans la liste de 'valid\_metrics' de Ball Tree ni de KD Tree.

=> KDTree.valid\_metrics : 'euclidean', 'l2', 'minkowski', 'p', 'manhattan', 'cityblock', 'l1', 'chebyshev', 'infinity'

=> BallTree.valid\_metrics : 'euclidean', 'l2', 'minkowski', 'p', 'manhattan', 'cityblock', 'l1', 'chebyshev', 'infinity', 'seuclidean', 'mahalanobis', 'wminkowski', 'hamming', 'canberra', 'braycurtis', 'matching', 'jaccard', 'dice', 'kulsinski', 'rogerstanimoto', 'russellrao', 'sokalmichener', 'sokalsneath', 'haversine', 'pyfunc'.

Remarque concernant les **metric** utilisées

Certaines 'metric' portent des noms différents mais font appel en réalité aux mêmes méthodes de calcul. Nos simulations permettent de bien le vérifier.

- "manhattan" : metrics.pairwise.manhattan\_distances
- "cityblock" : metrics.pairwise.manhattan\_distances
- "l1" : metrics.pairwise.manhattan\_distances
- "euclidean" : metrics.pairwise.euclidean\_distances
- "l2" : metrics.pairwise.euclidean\_distances
- "haversine" : metrics.pairwise.haversine\_distances
- "cosine" : metrics.pairwise.cosine\_distances
- "nan\_euclidean" : metrics.pairwise.nan\_euclidean\_distances
- "minkowski" : utilise la distance de Minkowski
- Pour la distance de Minkowsky, on peut résumer le calcul comme suit :

$$d(Echantillon, Voisin) = \left( \sum_{i=1}^n |Echantillon_i - Voisin_i|^p \right)^{1/p}$$

- Avec n, le nombre de dimensions, et p le facteur de puissance.
- p=2 => norme standard (distance euclidienne), valeur par défaut
- p=1 => distance de Manhattan
- p=infini => distance de "chebyshev"
- "chebyshev" : utilise la distance de Chebyshev

## Commentaires sur cette première simulation KNN

L'accuracy est de 84.52% > 77% du modèle "Désertique".

Par contre, le F1-Score (53%) et le Recall (40.9%) sont assez bas.

### 1. Simulations en faisant varier les principaux paramètres

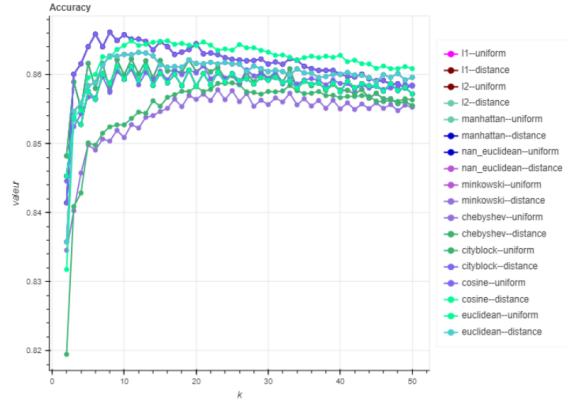
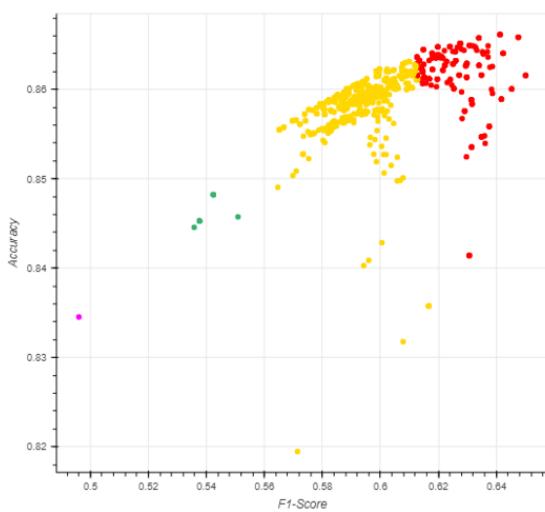
Avec toujours ce même jeu de données, nous allons faire varier dans nos simulations le nombre de voisins ainsi que le type de 'distance' utilisée et la pondération sur le calcul de la distance.

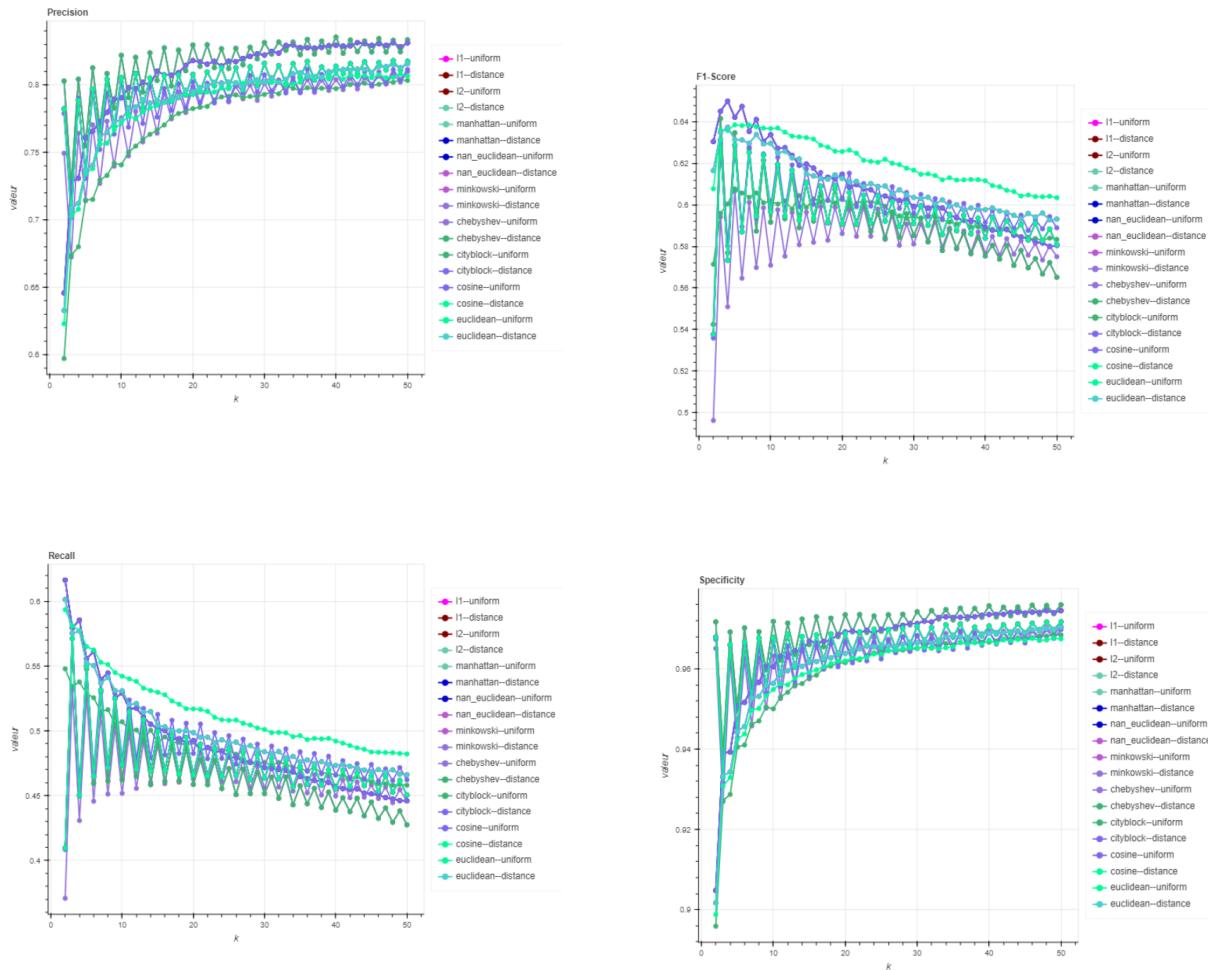
Paramètre	Plage de variations
<b>n_neighbors</b>	2 - 50
<b>metric</b>	'l1', 'l2', 'manhattan', 'nan_euclidean', 'minkowski', 'chebyshev', 'cityblock', 'cosine', 'euclidean'
<b>weights</b>	'uniform', 'distance'

algorithm = "auto", leaf\_size = 30, p = 2, metric\_params = None, n\_jobs = None

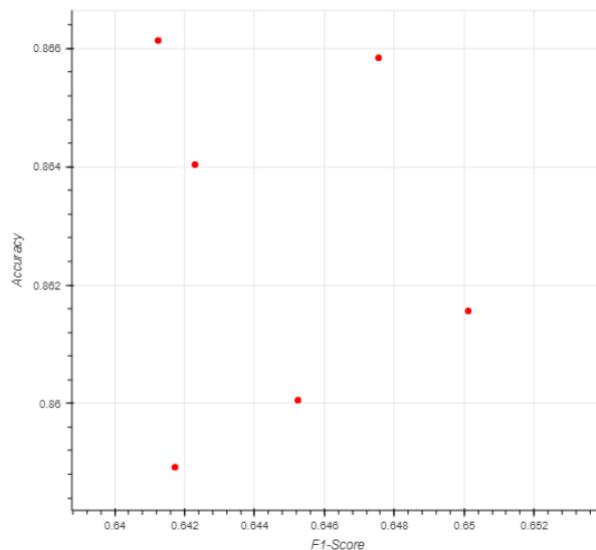
	Type	Valeur
0	Début traitement	2023-07-08 18:07:09.772407
1	Fin traitement	2023-07-09 01:35:32.542728
2	Durée traitement	7:28:22.770321
3	Fin sauvegarde	2023-07-09 01:36:14.307487
4	Durée sauvegarde	0:00:41.764759

Sous Bokeh, nous obtenons une représentation graphique de nos résultats :





Les meilleurs résultats sont ceux présentant la plus forte valeur de F1-score tout en gardant une Accuracy la plus importante. Il faut donc faire un compromis entre ces deux indicateurs. Nous choisissons 6 compromis parmi les meilleurs résultats.



Résultats	Paramètres de simulation
F1-Score : 0.641 Accuracy : 0.866 Recall : 0.545	$k = 8$ Metric = Manhattan ou L1 ou Cityblock Weights = Distance

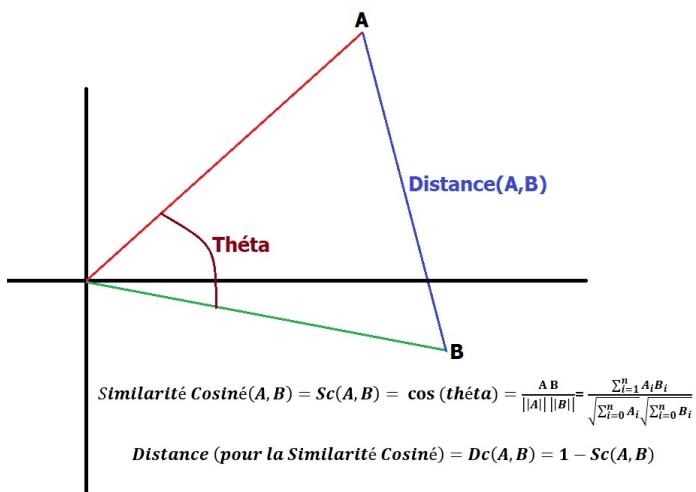
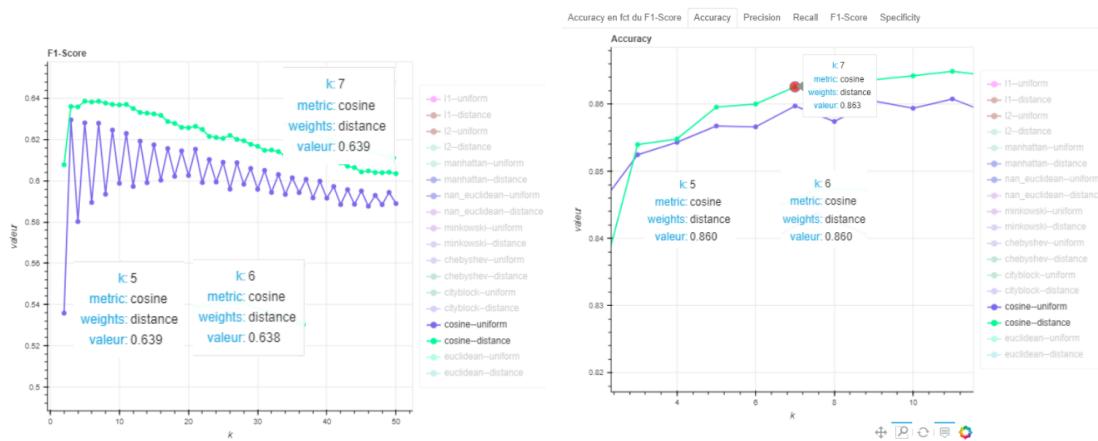
Precision : 0.780 Specificity : 0.957	
F1-Score : 0.648 Accuracy : 0.866 Recall : 0.561 Precision : 0.766 Specificity : 0.952	k = 6  Metric = Manhattan ou L1 ou Cityblock Weights = Distance
F1-Score : 0.642 Accuracy : 0.864 Recall : 0.556 Precision : 0.761 Specificity : 0.951	k = 5  Metric = Manhattan ou L1 ou Cityblock Weights = Distance
F1-Score : 0.650 Accuracy : 0.862 Recall : 0.586 Precision : 0.731 Specificity : 0.939	k = 4  Metric = Manhattan ou L1 ou Cityblock Weights = Distance
F1-Score : 0.645 Accuracy : 0.860 Recall : 0.579 Precision : 0.728 Specificity : 0.939	k = 3  Metric = Manhattan ou L1 ou Cityblock Weights = Distance
F1-Score : 0.642 Accuracy : 0.859 Recall : 0.575 Precision : 0.726 Specificity : 0.939	k = 3  Metric = Manhattan ou L1 ou Cityblock Weights = Uniform

L'Accuracy optimale est autour de 86% (Pour rappel, le **Modèle "Désertique"** qui prédit **qu'il ne pleut jamais** présente une accuracy de 77%...).

Ici, c'est donc le F1-Score qui sera notre principal indicateur.

## Métrique 'Cosine'

Les résultats obtenus avec cette métrique sont plus faibles qu'avec d'autres métriques.

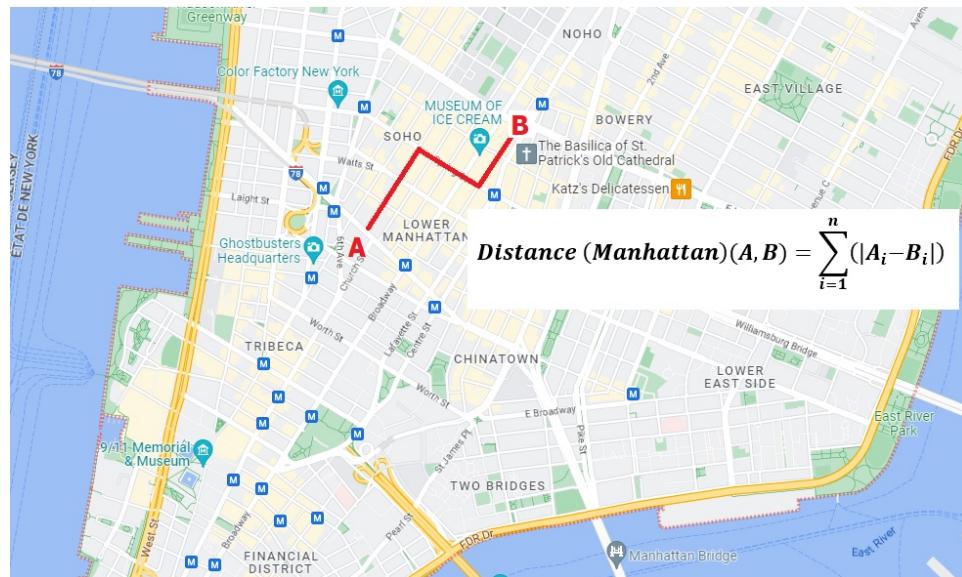


Plus les vecteurs pointent vers la même direction et plus les deux échantillons A et B sont proches.

Comme  $\cos(0) = 1$ , on prend la distance comme étant  $1 - \cos(\text{Théta})$ . La 'distance de cosiné' n'est pas une véritable distance, car l'inégalité triangulaire n'est pas vérifiée.

## Métrique 'Manhattan'

On fait la différence entre les 'coordonnées' de chaque échantillon.



Dans Scikit Learn, Les distances L1, Cityblock et Manhattan sont toutes trois associées à la "metric" définie par `metrics.pairwise.manhattan_distances`.

Ceci explique les résultats identiques. Nous avons donc effectué trois fois les mêmes simulations...

## 2. Analyse des résultats obtenus avec cette batterie de simulations

### La distance Manhattan obtient de meilleurs résultats

Prenons un exemple avec les deux variables 'Rainfall' et 'WindGustSpeed'. La normalisation MinMax ramène l'étendue des valeurs possibles sur l'intervalle [0,1]. Une même variation sur Rainfall normée et sur WindGustSpeed normée correspond en réalité à des variations très différentes sur les valeurs physiques non normées.

Lors du calcul des plus proches voisins, on utilise la différence entre deux valeurs de 'Rainfall' et deux valeurs de 'WindGustSpeed'. Ces deux différences ont la même importance pour calculer la distance entre voisins.

D'un point de vue physique, ces deux paramètres n'ont cependant pas du tout la même importance pour déterminer la valeur de 'RainTomorrow'. Et les différences sur ces deux paramètres n'ont pas non plus la même importance pour déterminer la valeur de RainTomorrow.

KNN n'est pas pensé pour prendre en compte la nature physique des différentes variables. C'est à nous de préparer les diverses variables pour permettre une comparaison "agnostique" sur la base de "metric".

- La distance euclidienne (ou minkowski (car  $p=2$ )) pose problème, car elle presuppose une relation "géographique" entre les relevés météo. Ce n'est globalement pas le cas.
- La distance cosiné presuppose également une relation "spatiale" entre les variables. Des relevés météo proches sont vus sous un angle solide proche. Ce n'est pas non plus le cas. De plus, le nombre important d'échantillons dans l'ensemble d'entraînement empêche l'apparition de directions privilégiées. Quelque soit la direction, on est proche d'un échantillon avec le label 1 et d'un échantillon avec le label 0.

- Enfin, la distance de Chebychev pose également problème. Cette distance suppose également une relation spatiale entre deux relevés météorologiques. On construit une “bulle” autour du relevé et les variables sont vues de manière équivalente. Ce n'est pas le cas.

**Il est donc logique que la “distance” qui donne le meilleur résultat soit une distance qui n'implique pas à priori une relation physique ou une “équi-pondération” entre les variables.**

Au final, la distance “Manhattan” reste la plus appropriée, car ce n'est justement PAS une distance, mais simplement un moyen pour indiquer la similarité entre deux relevés météo. Aucune hypothèse n'est faite sur l'équivalence entre telle ou telle autre variable.

#### **Cinq des 6 meilleurs résultats ont pour paramètre weight = Distance**

Ceci signifie que les relevés météorologiques impliquant de la pluie ou du temps sec le lendemain sont proches les uns des autres et que le seul moyen pour obtenir une bonne labellisation de l'échantillon est d'appliquer une pondération inversement proportionnelle à la distance.

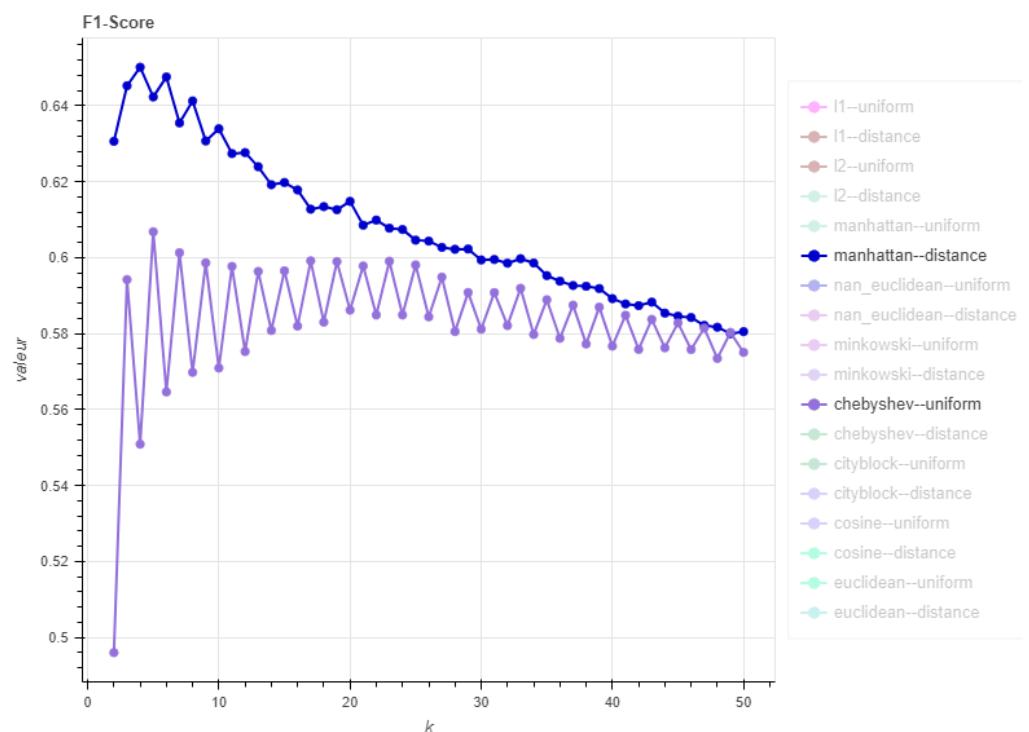
A partir de 3 voisins ou plus, les voisins supplémentaires apportent une perturbation sur le vote fourni par les trois voisins les plus proches. Cette perturbation est minimisée en ajoutant une pondération inversement proportionnelle à la distance trouvée.

#### **Les meilleurs résultats sont obtenus pour le nombre de voisin k (2< k < 9)**

L'essentiel du vote est apporté par les premiers voisins et au-delà de 8 voisins, les votes supplémentaires des voisins plus éloignés apportent plutôt une perturbation plutôt qu'un renforcement positif des résultats.

#### **Les courbes représentant les métriques en fonction de k présentent toutes une forme d'accordéon quand le paramètre weights vaut “uniform”**

Prenons l'exemple du F1-Score pour la metric Chebyshev / distance : uniform et la metric Manhattan /distance : distance



Si tous les voisins pris en compte ont le même poids, alors le fait de rajouter un voisin va déséquilibrer le vote. De plus, avec un nombre impair de voisins, le vote est facile à effectuer. Avec un nombre pair de voisins et en cas d'égalité des votes, le label de classe est choisi en fonction de l'ordre des classes dans les données d'entraînement. Par exemple, si la classe 0 apparaît en premier dans les données d'entraînement, le classifieur attribuera le label 0. Ceci explique la baisse de performance sur les valeurs paires pour Chebychev - uniform.

#### **Le F1-Score maximum ne dépasse pas 0.65 et 0.86 pour l'accuracy maximale**

Nous proposons deux explications ici:

- Les variables utilisées pour KNN sont mal adaptées à une comparaison via une "distance". Des variables parfaites pour KNN seraient pondérées entre elles en fonction directe de leur importance dans la détermination de la target et seraient comparables entre elles via le calcul de la distance. Ce n'est clairement pas facile d'imaginer de telles variables.
- Les variables actuellement utilisées ne contiennent pas "toute l'information permettant de déterminer la valeur de Raintomorrow".
- Nous manquons de données sur les évolutions nuageuses. Connaître les types de nuages apportant la pluie est salutaire. C'est le réflexe habituel quand une personne sort de chez elle : si le ciel est nuageux et présente des nuages menaçants, il vaut mieux prendre un parapluie.
- Nous manquons de données géographiques : un maillage du territoire avec un réseau plus dense de stations permettrait de connaître les endroits proches où il pleut et de suivre le déplacement des masses nuageuses apportant la pluie.
- La fréquence des relevés météorologiques est trop faible : Nous manquons de relevés météorologiques fréquents. Disposer d'une information mise à jour régulièrement permettrait de suivre les changements de manière plus fine.

### **3. Proposition d'amélioration: Création d'une 'metric' personnalisée**

#### **Création de "distance\_mi" basée sur l'information mutuelle**

**Contrainte technique :** KNeighborsClassifier permet de faire appel à une fonction 'metric' développée par nos soins. Cependant, cette fonction sera appelée à chaque calcul de distance. Il y aura très probablement des dizaines de milliers de calculs de distances. Comme ce code externe s'exécute en code interprété Python, il est potentiellement très lent. Nous tentons ici d'utiliser une 'metric' facile à calculer.

**Nous commençons par prendre la 'metric' Manhattan**, car c'est la '**metrics'** ayant donné les meilleurs résultats sur la simulation précédente. De plus, le code de cette distance est simple. On fait la différence entre les coordonnées des deux échantillons, puis on prend la valeur absolue et enfin on additionne ces valeurs absolues.

La distance de Manhattan est basée sur la norme "L1" (Taxicab norm) : somme des valeurs absolues des coordonnées.

L1 est bien une norme, car elle vérifie les trois propriétés :

- Séparation :  $N(x) = 0 \Leftrightarrow x = 0$
- Homogénéité :  $N(\lambda * x) = |\lambda| * N(x)$
- Inégalité triangulaire :  $N(x + y) \leq N(x) + N(y)$

La distance de Manhattan est créée à partir de cette norme et sera donc :

$$d(x,y) = N(x-y)$$

**Prise en compte d'une pondération** : les résultats de l'info mutuelle calculée sur l'ensemble d'entraînement utilisé pour entraîner le modèle (Par exemple 'X\_train\_minmax') permettent d'apporter un critère discriminant entre les variables. Plus l'information mutuelle sur une variable est grande et plus cette variable doit avoir de 'poids' sur le calcul de la distance.

Nous nous proposons donc simplement de multiplier cette pondération sur chaque différence de variable. Ceci nous permettra de calculer la distance "metric" personnalisée.

**Remarque** : Ceci revient donc exactement au même que d'appliquer une pondération en amont sur les variables du jeu de données. Mais dans notre cas, le calcul pondéré de la distance est effectué sur chaque couple ("échantillon", voisin).

La formule pour un calcul personnalisé "distance\_mi" sera donc :

"Pour les échantillons A et B, sur la dimension i :  $\text{abs}(A(i) - B(i)) * \text{info mutuelle}(i)$ "

### **Simulation avec la distance "distance\_mi":**

Comme vu plus haut, nous utilisons les paramètres :

- leaf\_size = 30 : propose un bon compromis
- weights = 'distance'
- metric = distance\_mi
- algorithm = 'auto'.
- k = 4 : Afin de limiter les calculs et en utilisant la distance personnalisée "distance\_mi", nous choisissons k = 4 qui a donné de bons résultats avec la distance de Manhattan.

Est-ce que 'brute force' sera utilisé ?

- 'peu de données' ? ==> **faux**,
- metric = 'precomputed' ? ==> **faux**,
- 'D>15' ? ==> **faux**
- '(D=15), 'k>N/2' ? ==> **faux**,
- le metric utilisée n'est pas dans la liste des métriques valides pour kd\_tree ou ball\_tree ? ==> **VRAI**.

Donc **'brute force'** sera utilisé.

La 'metric' utilisée est une fonction que nous avons créée. Donc l'algorithme retenu par KNN sera automatiquement 'brute force'.

D'un point de vue technique, la fonction accepte en entrée deux vecteurs (Series) dont les coordonnées sont les variables (colonnes) de X\_train ou de X\_test. La fonction renvoie un scalaire (float).

### **Analyse des résultats : "distance\_mi"**

Les résultats obtenus sont les suivants :

Metric	Valeur trouvée
accuracy	<b>0.8469965999244428</b>

precision	0.6932814021421616
recall	0.5442384865278043
f1-score	<b>0.6097848196124611</b>
specificity	0.9322216245293168

Le f1-score obtenu n'est pas noté parmi les meilleurs.

#### Durée du traitement importante :

	Type	Valeur
0	Début traitement	2023-07-16 03:17:16.433223
1	Fin traitement	2023-07-16 04:38:23.958088
2	Durée traitement	<b>1:21:07.524865</b>

Comme pressenti , la durée du traitement pour k= 4 est de 1h21 minutes ce qui est tout à fait prohibitif.

Nous ne recommandons donc pas d'utiliser une fonction personnalisée pour le calcul des distances pour Kapy.

#### Usage “détourné” d'une fonction personnalisée ?

Le classifieur KNeighborsClassifier utilisera l'algorithme “Brute Force” à chaque fois que la distance sera une distance personnalisée. Aussi nous recommandons de réserver l'usage d'une fonction personnalisée uniquement pour les petits jeux de données ou pour des ensembles de paramètres déjà optimisés.

En effet, utiliser une fonction personnalisée pour rechercher les meilleurs jeux de paramètres va provoquer une véritable explosion des temps de calcul pour la simulation.

On peut penser également à l'usage d'une fonction personnalisée en “détournant” l'utilisation du KNeighborsClassifier. En effet, la “Brute force” va amener le classifieur à effectuer un ensemble de calculs par échantillon de l'ensemble de test. Cet ensemble de calculs contiendra l'ensemble des échantillons de l'ensemble d'entraînement.

On aura donc au total N \* M appels à la fonction personnalisée (N échantillons dans l'ensemble de test , M échantillons dans l'ensemble d'entraînement).

Enfin, cette fonction personnalisée n'a pas besoin d'être un calcul de distance. Par contre elle doit prendre en entrée un échantillon du jeu de test et un échantillon du jeu d'entraînement et renvoie un scalaire.

#### Autres usages d'une fonction personnalisée :

Dans le cadre de Kapy, l'usage de ce classifieur avec une fonction personnalisée n'est pas adapté. Par contre on peut imaginer d'autres cas où cette fonction pourrait être utilisée. En recherchant dans la littérature, nous avons trouvé ces deux autres exemples :

Autre fonction	Définition	Usage
Dynamic Time Warping	from fastdtw import fastdtw	Travail sur des séries

(Distance dynamique temporelle)	<pre>def dtw_distance(x, y):     distance, _ = fastdtw(x, y)     return distance</pre>	temporelles  <a href="https://hal.science/hal-00647522/document">https://hal.science/hal-00647522/document</a>
Distance de Jaccard	<pre>from scipy.spatial import distance  def jaccard_distance(x, y):     return distance.jaccard(x, y)</pre>	La distance de Jaccard peut être utilisée dans le traitement des données textuelles et la recommandation d'articles.  <a href="https://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard">https://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard</a>  <a href="https://www.arboretum.link/notes/similarit%C3%A9">https://www.arboretum.link/notes/similarit%C3%A9</a>
Distance pondérée par les variables (comme la tentative précédente)	<pre>def weighted_distance(x, y):     weights = [a, b, c, ...]     # Poids respectifs des attributs     squared_diff = (x - y) ** 2     weighted_squared_diff = squared_diff * weights     return np.sqrt(np.sum(weighted_squared_diff))  ou encore dans le cas de Kapy :  def distance_mi_inv(x,y):     ma_distance =     np.abs(x-y).dot(mi_inv)     # mi_inv : inv. de l'info mutuelle.     return ma_distance</pre>	Si certaines variables sont plus importantes que d'autres dans la classification, nous pouvons définir une distance qui va attribuer des poids (weights = [a, b, c, ...]) différents à ces variables lors du calcul de la distance.  ⇒ notre projet Kapy et le KNeighborsClassifier

#### 4. Proposition d'amélioration: Création d'une fonction de pondération

##### Création d'une fonction de pondérations "weights\_sqr" pour le paramètre weights

Nous avons vu un premier usage d'une fonction personnalisée pour calculer la distance entre deux points. Nous nous proposons maintenant de concevoir une fonction afin d'augmenter la capacité à différencier les voisins entre eux. La fonction 'distance' propose d'appliquer un ratio inversement proportionnel à la distance calculée.

Nous allons tenter ici de définir une pondération qui ne varie pas en  $1/x$  mais en  $1/x^2$ .

Puis nous ferons une seconde simulation avec une pondération en  $1/x^2 * 1/x^2$ .

Et enfin une troisième simulation avec une pondération en  $1/x^2 * 1/x^2 * 1/x^2 * 1/x^2$ .

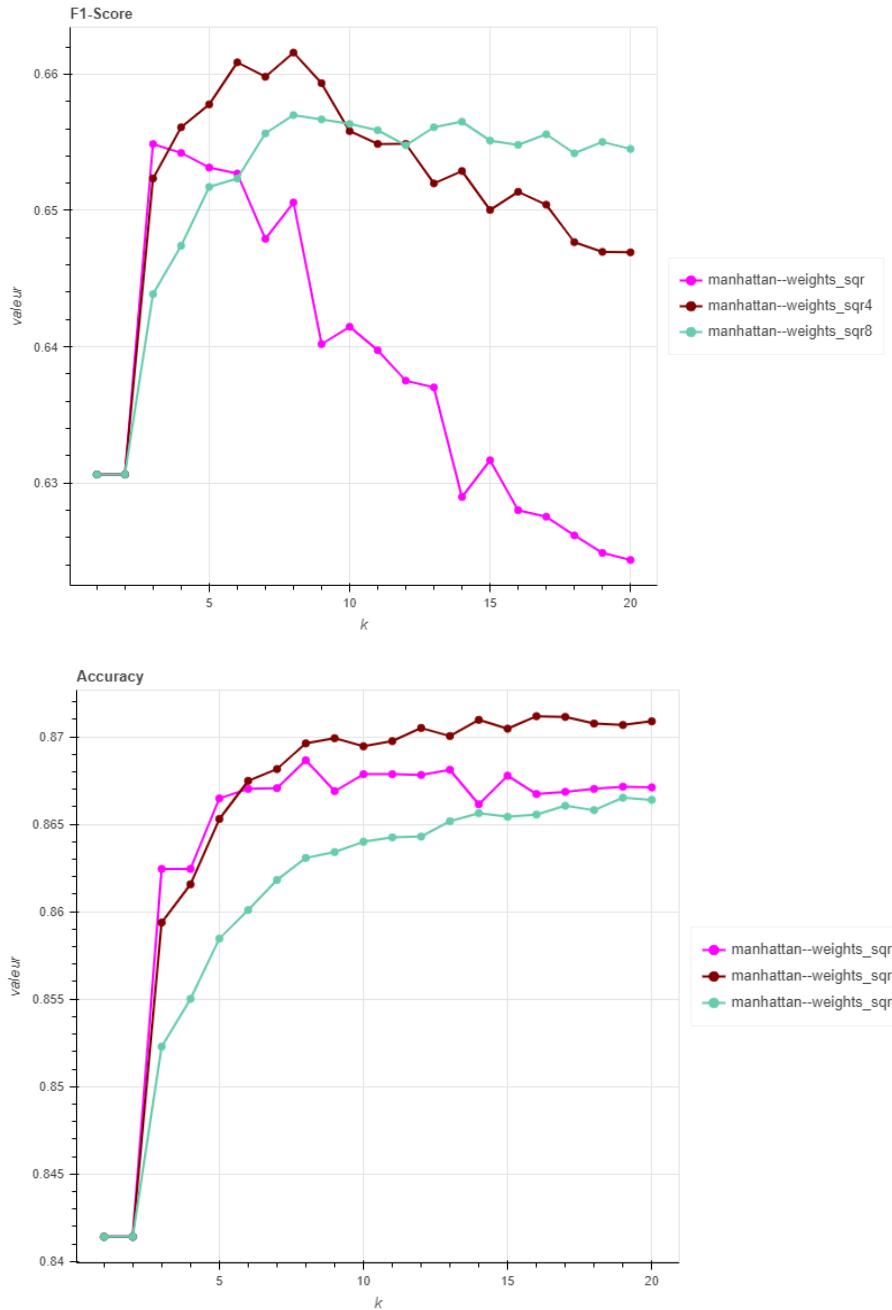
D'un point de vue technique, la fonction reçoit en entrée un tableau à une dimension contenant des distances et retourne un tableau ayant les mêmes dimensions et contenant les distances pondérées.

## Considérations techniques :

La taille du tableau passé en paramètres varie en fonction de k. L'utilisation des divers algorithmes permettant d'éviter l'utilisation de "brute force" entraîne une variation de la taille du tableau.

Par ailleurs, l'appel à la fonction de pondération ne modifie que de façon très marginale la durée du traitement.

## Résultats :



Le meilleur compromis (f1-score, accuracy) est obtenu avec le couple (f1-score de 0.662, accuracy de 0.870) pour k= 8 et pour la fonction de pondération "weights\_sqr4".

L'utilisation de la fonction de pondération permet d'améliorer légèrement les résultats : f1-score augmenté de 1.2% et l'accuracy augmentée de 0.4%.

Résultats	Paramètres de simulation
<b>F1-Score : 0.662</b> <b>Accuracy : 0.870</b> Recall : 0.580 Precision : 0.770 Specificity : 0.951	k = 8 Metric = Manhattan Weights = "weights_sqrt4"

L'utilisation d'une distance plus discriminante permet justement de différencier des voisins proches de l'échantillon.

##### 5. Proposition d'amélioration: Modifier le jeu de données en ajoutant une pondération des variables via l'information mutuelle

Il faudrait donc pouvoir **rajouter un coefficient sur chaque variable en fonction de sa contribution** au moment de déterminer la distance. La pondération pourrait par exemple prendre en compte les résultats de divers tests :

- **Le F-tests statistics** qui calcule la valeur absolue de la corrélation (au sens de Pearson). On obtient un score entre 0 et 1. Il s'agit d'une indication qu'il existe une relation de linéarité entre la variable et la target 'Raintomorrow'.

ou encore

- **La mutual information** qui calcule l'information mutuelle entre la variable et 'RainTomorrow'. Ce test compare les distributions de probabilité de la variable avec celle de la target 'RainTomorrow'. Cet indicateur est très intéressant car il fait abstraction des relations linéaires et prend en compte l'ensemble de l'information partagée. Enfin, cet indicateur est implémenté dans SciKit-Learn en utilisant... la notion de plus proches voisins (NearestNeighbors, avec 3 voisins). ref : [https://github.com/scikit-learn/scikit-learn/blob/364c77e04/sklearn/feature\\_selection/\\_mutual\\_info.py#L312](https://github.com/scikit-learn/scikit-learn/blob/364c77e04/sklearn/feature_selection/_mutual_info.py#L312)

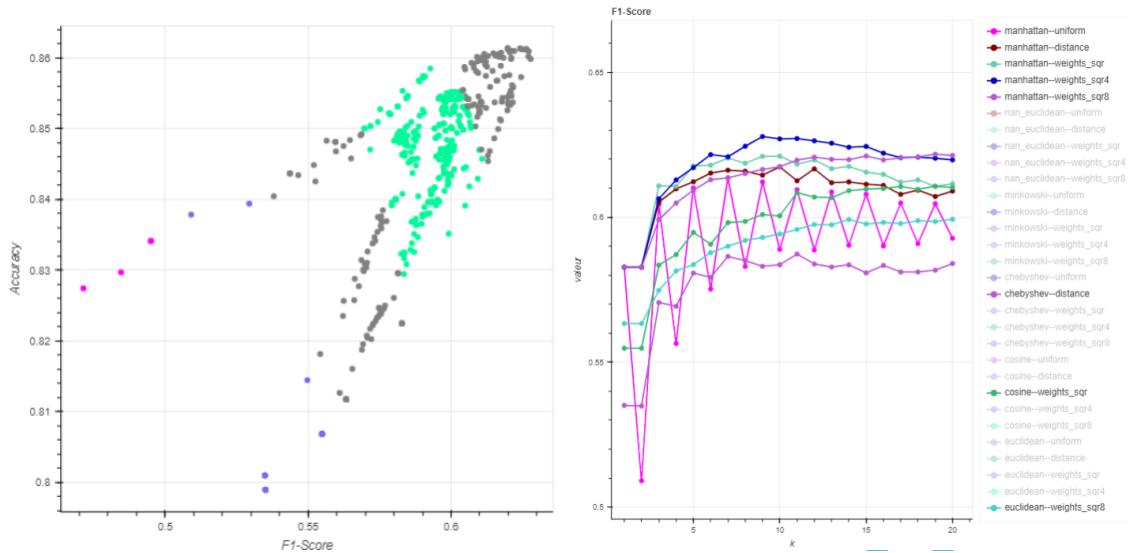
En regardant le cumul de l'information mutuelle, on obtient 95% de toute l'information mutuelle avec ces 15 variables:

No de la variable	Valeur de l'info. Mutuelle	Nom variable
13	0.119809	Humidity
6	0.188815	diffTempMinMax
0	0.243969	Rainfall
15	0.292354	DeltaH_2d
16	0.339861	DeltaH_3d
17	0.386933	WindDirInfluence
14	0.431162	DeltaH_1d
18	0.474591	consecutiveRainingDays
12	0.513474	diffHumidity9am3pm

8	0.540965	Pressure
1	0.567500	WindGustSpeed
11	0.592497	DeltaP_3d
10	0.615595	DeltaP_2d
9	0.633429	DeltaP_1d
7	0.642716	diffPressure9am3pm

Nous effectuons notre nouvelle simulation en ajoutant une pondération sur le jeu de données basée sur la “Mutual Information”.

### Analyse des résultats de la pondération des variables via l'information mutuelle



Résultats	Paramètres
F1-Score : 0.627 Accuracy : 0.861 Recall : 0.532 Precision : 0.763 Specificity : 0.953	k = 11 Metric = manhattan Weights = weights_sqr4
F1-Score : 0.628 Accuracy : 0.860 Recall : 0.538 Precision : 0.753 Specificity : 0.950	k = 9 Metric = manhattan Weights = weights_sqr4

**La pondération sur la base de l'Information Mutuelle ne permet pas d'améliorer les résultats.**

#### iv. Utilisation d'une PCA sur KNN

##### Principe de la PCA

L'idée derrière l'analyse en Composantes Principales (PCA) est de considérer que chaque variable est une dimension d'analyse et que cette dimension contient un partie de l'information permettant de prédire la valeur de la cible.

La PCA est une technique qui permet d'extraire l'essentiel de l'information contenue dans ces  $n$  dimensions du jeu de données initial. Cette information est alors répartie sur un ensemble de ' $m$ ' composantes principales (' $m < n$ ') qui forment un nouveau jeu de données.

Grâce à cette transformation, en perdant un minimum d'information, on peut gagner un temps considérable pour la modélisation.

La technique mathématique sous-jacente consiste à trouver des transformations pour maximiser l'orthogonalité entre les composantes (des composantes qui ne sont pas corrélées linéairement). On utilise pour cela le concept de "variance expliquée". La variance expliquée par une composante principale est équivalente à la "variance des données projetées sur cette composante".

##### a.4.1.1) Pourquoi utiliser une PCA avec KNeighborsClassifier ?

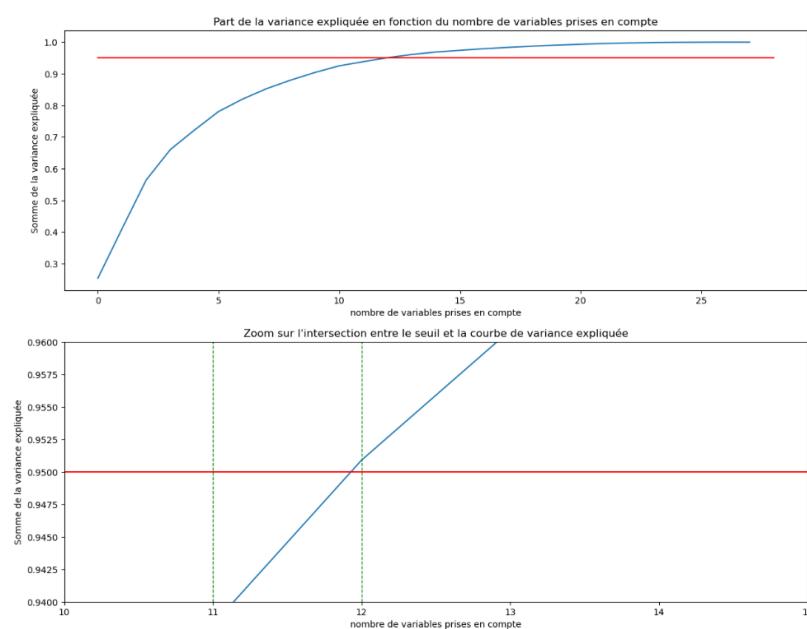
La durée des traitements pour KNN peut être assez longue. En utilisant un ePCA sur KNN on pourra donc raccourcir d'autant la durée de traitement et tenter plus de scénarios de simulation.

Ensuite, la PCA modifie les variables physiques initiales en créant des composantes sans valeur physique. Ceci s'adapte bien à KNeighborsClassifier, car cet algorithme, en particulier avec la distance de Manhattan, compare entre elles des variables de nature physique différentes. Ceci n'a aucun sens physique.

Alors autant abandonner tout espoir de relier l'algorithme KNeighborsClassifier à une explication physique sur les variables et concentrons-nous sur des algorithmes permettant de répartir la variance de l'information d'origine sur des axes indépendants.

##### a.4.1.1.2) Recherche du nombre optimal de composantes à utiliser

Nous fixons à 95% de variance expliquée le seuil pour déterminer le nombre de composantes PCA à garder.



On prendra donc les 12 premières composantes de la PCA.

## 1. Simulation PCA sur 12 composantes principales

### Nombre de composantes retenues

Nous allons garder les 12 composantes principales qui représentent 95% de la variance expliquée (à comparer aux 15 variables présentant 95% de l'information mutuelle).

### Analyse des résultats

Nous effectuons une série de simulation sur ce jeu de données avec les paramètres suivants :

Paramètres	Valeurs possibles
metric	Manhattan, Pond_PCA*
weights	uniform, distance, weights_sqr, weights_sqr4, weights_sqr8
k	1 à 20

Pour les autres paramètres, les valeurs sont :

algorithm = "auto", leaf\_size = 30, p = 2, metric\_params = None, n\_jobs = None.

#### \*: Remarque concernant Pond\_PCA

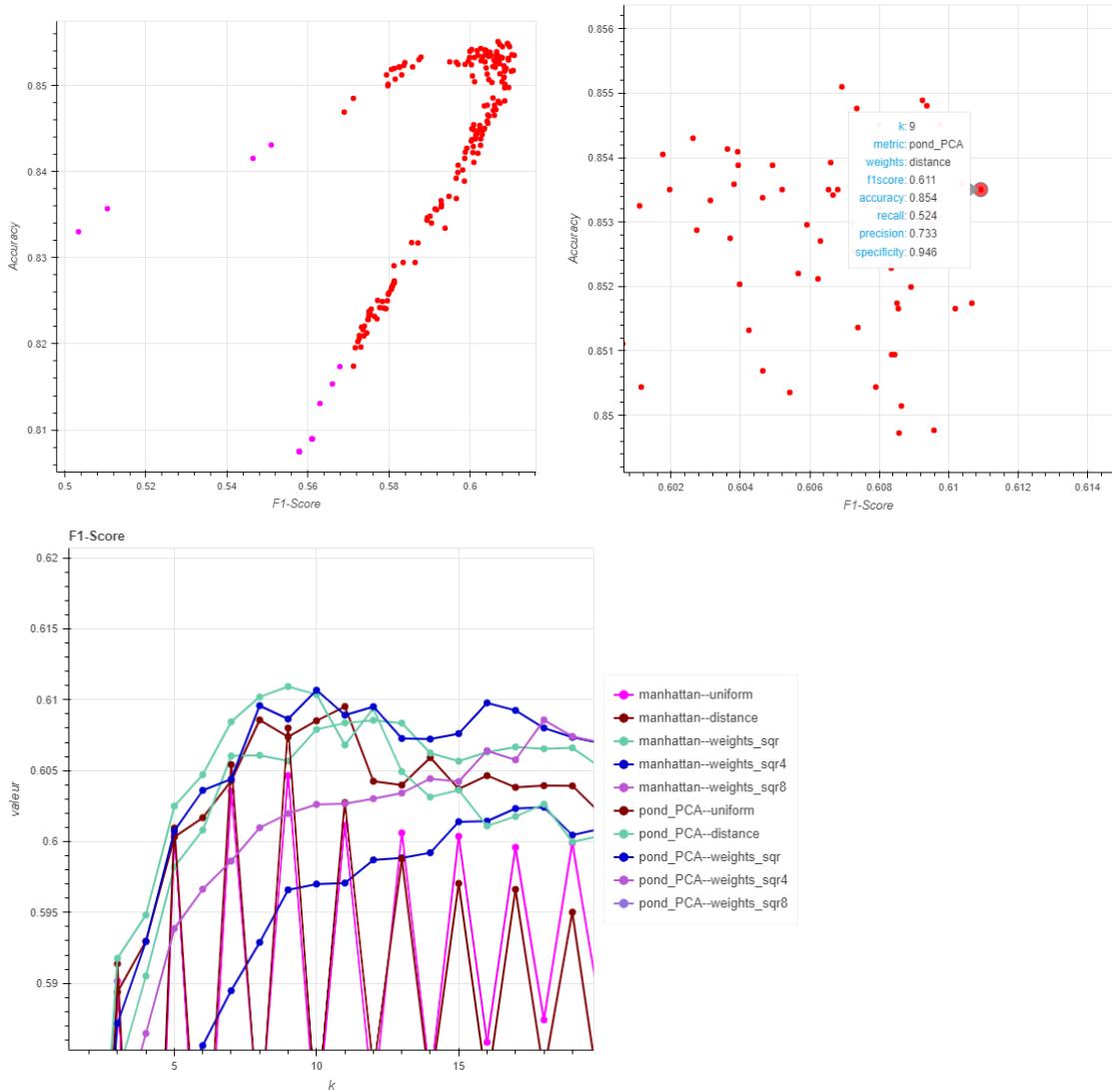
On a défini une fonction de pondération qui applique simplement une pondération égale à la variable expliquée de chaque variable. Puis nous effectuons simplement un calcul de distance de Manhattan. Ceci revient donc à effectuer un travail préalable sur le jeu de données que l'on traite en appliquant une pondération sur chacune de ses variables. On gagne ainsi énormément de temps plutôt que d'appeler la fonction pour chaque calcul de distance.

### Nous obtenons les résultats suivants :

	Type	Valeur
0	Début traitement	2023-07-16 02:21:20.859391
1	Fin traitement	2023-07-16 02:26:08.867338
2	Durée traitement	0:04:48.007947
3	Fin sauvegarde	2023-07-16 02:26:14.988339
4	Durée sauvegarde	0:00:06.121001

Quatre minutes pour un ensemble de simulations. La PCA permet vraiment d'abaisser la durée des simulations en diminuant le nombre de variables.

### Scores Accuracy et F1-Score :



Nous notons les deux meilleurs scores obtenus et nous rappelons en dessous le meilleur score obtenu précédemment.

Résultats	Paramètres de simulation
<b>F1-Score : 0.611</b> <b>Accuracy : 0.854</b> Recall : 0.524 Precision : 0.733 Specificity : 0.946	k = 9 Metric = <b>pond_PCA</b> Weights = <b>distance</b>
<b>F1-Score : 0.610</b> <b>Accuracy : 0.855</b> Recall : 0.517 Precision : 0.742 Specificity : 0.949	k = 16 Metric = <b>pond_PCA</b> Weights = <b>weights_sqr</b>
<b>F1-Score : 0.662</b>	k = 8

**Accuracy : 0.870**

Recall : 0.580

Precision : 0.770

Specificity : 0.951

Metric = Manhattan

Weights = "weights\_sqr4"

Les meilleures performances obtenues avec la PCA sont inférieures aux meilleures performances obtenues avec les autres simulations. Ceci est normal, car on a perdu 5% de l'information en réduisant à 12 composantes les variables.

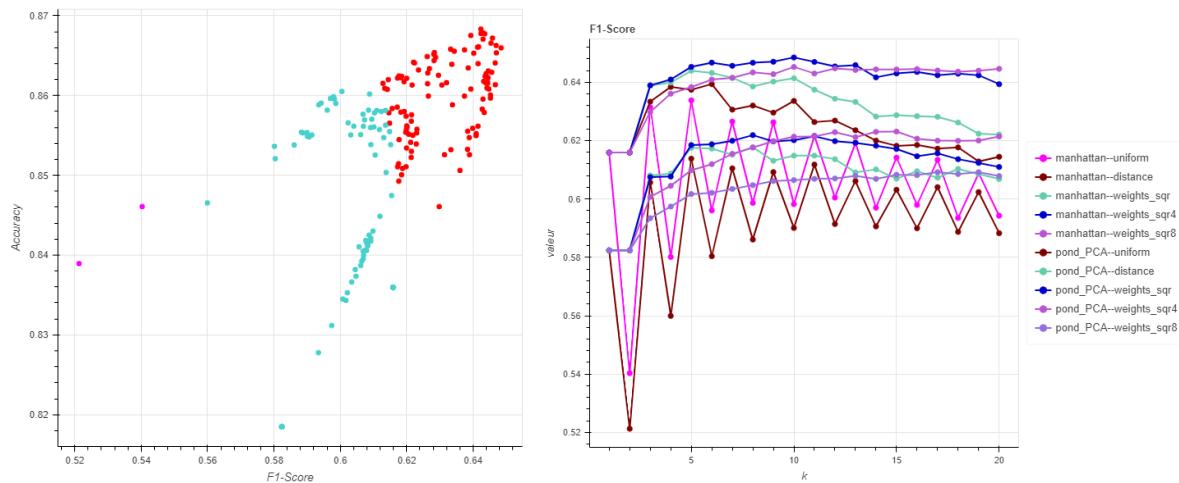
Par contre, on peut observer qu'une pondération basée sur la variance expliquée permet d'obtenir le meilleur score des simulations avec cette PCA.

## 2. Simulation PCA sur les 28 composantes principales (sans pondération)

Nous effectuons deux nouvelles simulations basées sur la PCA, mais sans faire de réduction de dimension. En effet, avec 95% de la variance expliquée ceci entraîne 5% de perte d'information.

Par ailleurs, nous faisons l'hypothèse que l'utilisation des composantes principales permet de ne plus faire référence à des variables physiques. Leur comparaison dans le calcul d'une distance fait alors sens. Nous allons voir ce qu'il en est.

### PCA sans pondération sur les 28 composantes



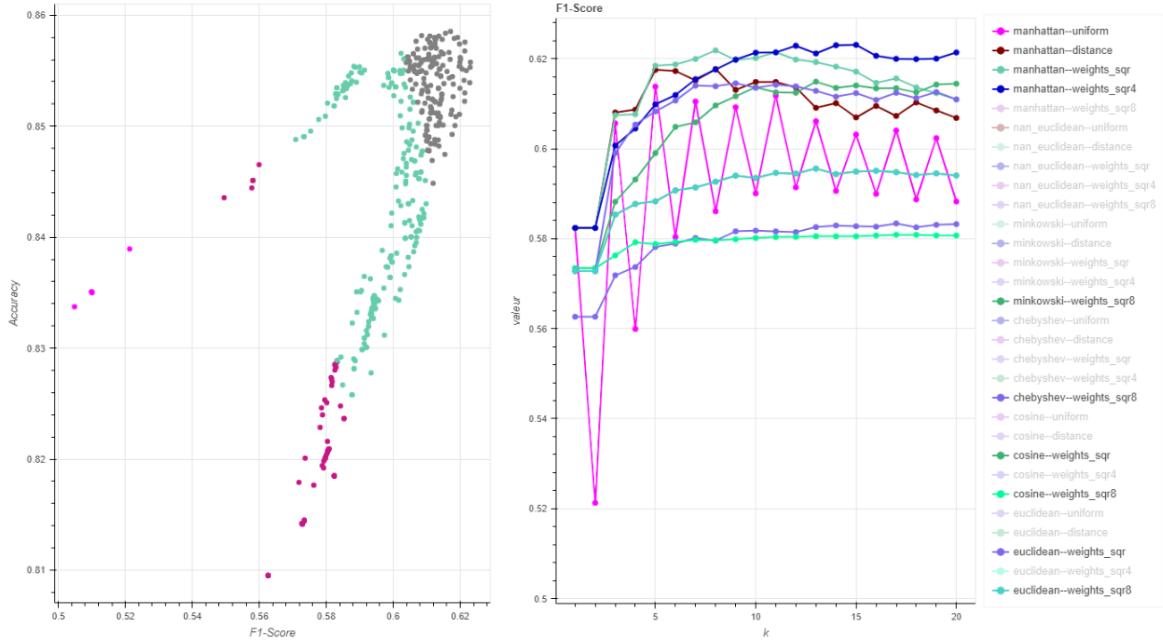
Le meilleur compromis semble obtenu pour :

- k : 10
- metric : manhattan
- weights : weights-sqr4
- f1-score : 0.648
- accuracy : 0.866
- recall : 0.563
- precision : 0.765
- specificity : 0.951

L'utilisation d'une PCA ne permet pas d'améliorer les résultats précédents. KNeighborsClassifier n'est pas particulièrement sensible à cette transformation.

### 3. Simulation PCA sur les 28 composantes principales (avec pondération)

#### PCA pondérée sur les 28 composantes



Le meilleur compromis entre f1-Score et Accuracy semble être obtenu pour :

- $k : 15$
- metric : manhattan
- weights : weights\_sqr4
- f1-score : 0.623
- accuracy : 0.856
- recall : 0.543
- precision : 0.731
- specificity : 0.944

Ce compromis est toutefois moins bon qu'avec la PCA non pondérée.

#### 4. Conclusion sur l'utilisation de la PCA avec KNeighborsClassifier

L'utilisation de la PCA n'apporte pas de plus-value pour KNeighborsClassifier dans le cadre de Kapy. Nous devons donc continuer à utiliser la PCA dans son cadre habituel : une réduction de dimension tout en gardant un maximum de la variance expliquée. Ceci permet de diminuer de manière très notable la durée des simulations, au prix toutefois d'une performance amoindrie.

## v. Utilisation de KNeighborsClassifier sur un KMeans

### Principe du KMeans

L'algorithme KMeans est une manière élégante de déterminer k centroïdes à partir d'un ensemble d'échantillons. La valeur k est fournie en paramètre.

Voici les étapes suivies par l'algorithme :

- 1. L'algorithme va déterminer de manière aléatoire k centroïdes pour initialiser la construction des k clusters
- 2. Un calcul de distance est effectué entre chaque échantillon et chacun des k centroïdes. Chaque échantillon est alors affecté au cluster le plus proche.
- 3) Un nouveau centroïde est calculé comme étant la résultante de la moyenne de tous les échantillons appartenant au même cluster.
- 4) Répéter les étapes 2 et 3 jusqu'à ce que le calcul de cluster devienne stable ou bien jusqu'à ce que le nombre d'itération (fourni en paramètre, par défaut : 300).
- 5) Calculer la variance observée au sein de chaque cluster (somme des variances)
- 6) Répéter les étapes 1 à 5 jusqu'à atteindre un minimum de somme de variances

L'algorithme va donc se stabiliser sur des minima de variance, mais ces minima ne sont pas automatiquement des minima absolus de variance..

### Simulations effectuées

Nous allons utiliser KMeans pour diminuer le nombre d'échantillons de l'ensemble d'entraînement et rééquilibrer le jeu de données.

A chaque fois, nous créons un jeu de centroïdes composés de deux ensembles de centroïdes :

- A partir des échantillons d'entraînement avec le label 0, nous créons un ensemble de centroïdes avec le label 0 (pas de pluie demain).
- A partir des échantillons d'entraînement avec le label 1, nous créons un ensemble de centroïdes avec le label 1 (pluie demain).

En rassemblant ces deux groupes de centroïdes, on forme le nouvel ensemble d'entraînement. Nous pouvons donc jouer à la fois sur le nombre total de centroïdes mais aussi sur le ratio entre chaque label. Nous pouvons ainsi rééquilibrer le jeu de données.

Pour chaque simulation, nous faisons varier les trois principaux paramètres :

- k, le nombre de voisins varie entre 1 et 20,
- metric : toutes les métriques proposées en standard
- weights : uniform et distance plus les 3 fonctions de pondération personnalisées : 'weights\_sqr', 'weights\_sqr4' et 'weights\_sqr8'.

Nous pouvons nous permettre de jouer toutes ces simulations, car l'un des principaux intérêts de KMeans est de réduire la taille des échantillons d'entraînement, tout en conservant dans chaque centroïde (nous l'espérons) le pattern des échantillons composant son cluster.

## Analyse des Résultats

Nous résumons les principaux résultats obtenus dans le tableau ci-dessous. A chaque fois, nous choisissons "visuellement" la simulation proposant le meilleur compromis entre Accuracy et F1-Score.

Simulation	Résultats	Paramètres	Remarque								
KNeighborsClassifier sur 500 centroïdes sec et 500 centroïdes pluie	<b>F1-Score : 0.662</b> <b>Accuracy : 0.833</b> Recall : 0.742 Precision : 0.597 Specificity : 0.859	k = 15 Metric = euclidean Weights = weights_sqrt4	Meilleur f1-score de toutes nos simulations  <b>La metric manhattan sous-performe</b>								
KNeighborsClassifier sur 400 centroïdes sec et 600 centroïdes pluie	<b>F1-Score : 0.634</b> <b>Accuracy : 0.816</b> Recall : 0.726 Precision : 0.563 Specificity : 0.842	k = 6 Metric = euclidean Weights = uniform	les metrique								
KNeighborsClassifier sur 200 centroïdes sec et 800 centroïdes pluie	<b>F1-Score : 0.606</b> <b>Accuracy : 0.800</b> Recall : 0.702 Precision : 0.534 Specificity : 0.827	k = 2 Metric = cosine Weights = uniform									
KNeighborsClassifier sur 600 centroïdes sec et 400 centroïdes pluie	<b>F1-Score : 0.660</b> <b>Accuracy : 0.855</b> Recall : 0.642 Precision : 0.679 Specificity : 0.914	k = 14 Metric = cosine Weights = weights_sqrt4	cosine : il existe des directions privilégiées pour les clusters  équilibrage du jeu de données : un léger rééquilibrage permet d'améliorer les résultats								
KNeighborsClassifier sur 50 centroïdes sec et 50 centroïdes pluie	<b>F1-Score : 0.631</b> <b>Accuracy : 0.807</b> Recall : 0.750 Precision : 0.545 Specificity : 0.824	k = 8 Metric = euclidean Weights = weights_sqrt4									
KNeighborsClassifier sur 15 centroïdes sec et 15 centroïdes pluie	<b>F1-Score : 0.6.2</b> <b>Accuracy : 0.789</b> Recall : 0.727 Precision : 0.513 Specificity : 0.806	k = 7 Metric = Manhattan Weights = weights_sqrt4	Pour le calcul "k=13--metric=chebyshev --weights=uniform", on se retrouve avec une matrice de confusion totalement déformée  <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;"><b>Prédiction</b></td> <td style="text-align: center;"><b>1</b></td> </tr> <tr> <td style="text-align: center;"><b>Realité</b></td> <td></td> </tr> <tr> <td style="text-align: center;">0.0</td> <td style="text-align: center;">18590</td> </tr> <tr> <td style="text-align: center;">1.0</td> <td style="text-align: center;">5233</td> </tr> </table>	<b>Prédiction</b>	<b>1</b>	<b>Realité</b>		0.0	18590	1.0	5233
<b>Prédiction</b>	<b>1</b>										
<b>Realité</b>											
0.0	18590										
1.0	5233										

KNeighborsClassifier sur 100 centroïdes sec et 100 centroïdes pluie	<b>F1-Score : 0.649</b> <b>Accuracy : 0.830</b> Recall : 0.714 Precision : 0.594 Specificity : 0.863	k = 6 Metric = euclidean Weights = weights_sqrt	
---	--	---	--

L'utilisation de KMeans remet à l'ordre du jour l'utilisation de certaines métriques :

- nan\_euclidean : permet de gérer la présence de Nan. Sinon, on a exactement le même calcul de distance que pour la metric "euclidean"
- minkowski : comme p=2, comme vu précédemment, c'est la même distance que la distance euclidienne.
- euclidean : distance euclidienne classique
- cosine : permet de mettre en évidence des directions privilégiées pour la présence de centroïdes. Comme il y a beaucoup moins de points, certains secteurs de l'espace sont désormais vides. Cosine devient alors pertinent.

L'utilisation de centroïdes simplifie les calculs pour KNeighborsClassifier tout en gardant un pattern pluie/sec qui permet d'obtenir dans certains cas de bons résultats.

## vi. Conclusions sur l'utilisation du KNeighborsClassifier

Nos diverses tentatives pour améliorer les résultats de KNeighborsClassifier ne nous ont pas permis de nous détacher significativement des résultats obtenus avec les métriques proposées en standard et avec la pondération proposée en standard (paramètre "weights"). Des gains en performance ont toutefois pu être réalisés sans pour autant dépasser les quelques % d'amélioration sur le f1-score.

Une transformation des données via PCA ou encore la pondération entre les variables n'a pas permis d'améliorer significativement les performances. Sur ces jeux de données, la distance Manhattan est la plus performante.

L'utilisation d'une fonction de pondération personnalisée a permis de gagner un peu sur le f1-score.

La métrique personnalisée doit hélas être exclue en raison de l'augmentation des temps de calcul sur le dataset initial.

Enfin, l'utilisation de KMeans a permis de fortement diminuer les durées de traitement, tout en maintenant sur certains jeux de paramètres de bons scores. La distance de Manhattan sous-performe lorsque l'on a affaire à des centroïdes en nombre limité. On retrouve alors les metric "cosine" et "euclidean".

Enfin, avec KNNclassifier, il apparaît obligatoire de passer en revue et de simuler de manière agnostique de nombreuses combinaisons de paramètres. C'est ce que nous avons fait.

KNeighborsClassifier permet d'obtenir au final des résultats intéressants mais son amélioration se heurte à son absence de lien avec la réalité physique des variables manipulées. La seule voie d'amélioration restante sera comme à chaque fois de construire de nouvelles variables portant un supplément d'information sur la cible.

## b. Classification binaire avec rééchantillonage

Nous avons observé que notre variable cible présente une distribution déséquilibrée (~80/20).

Il est ici étudié des modèles de classification avec un ré-échantillonage préalable; soit par under-sampling, soit par over-sampling.

Les résultats sont résumés dans le tableau ci-après et font apparaître un léger gain grâce à la fonction de RandomUnderSampling et un ré-échantillonage en conservant un léger déséquilibre entre les 0 et les 1.

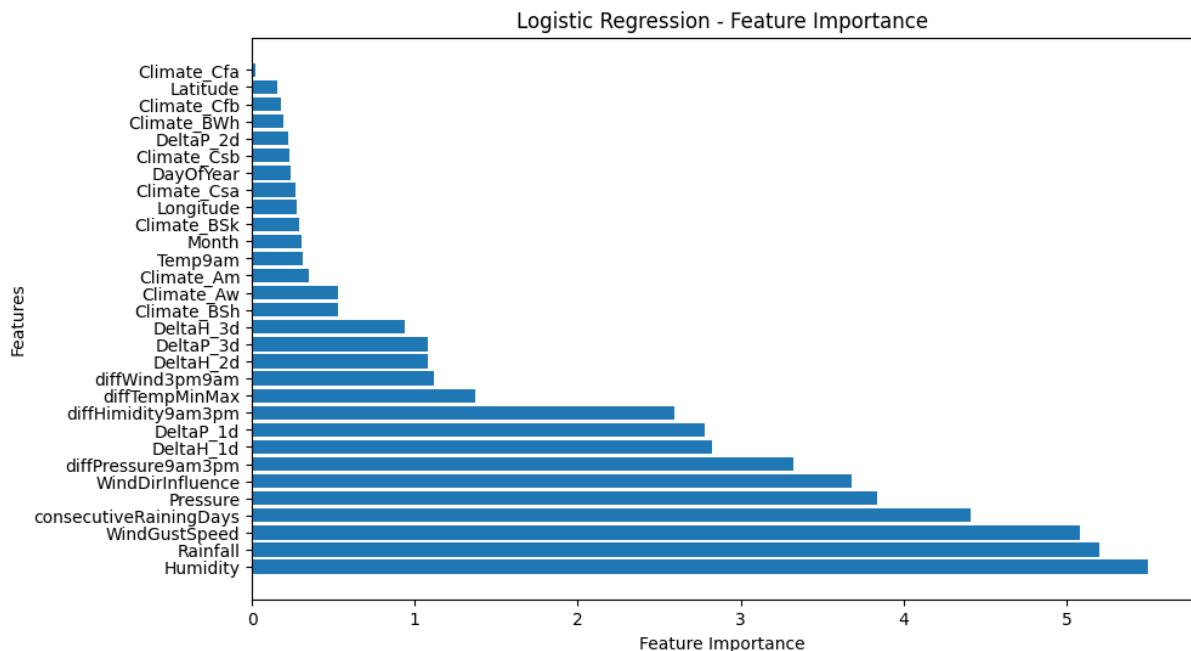
On note également que le modèle RandomForest fournit systématiquement les meilleurs scores.

Modèle	Pre-processing	f1-score	accuracy
LogisticRegression	-	0.62	0.85
LogisticRegression	RandomUndersampling 50/50	0.64	0.81
KNN	RandomUndersampling 50/50	0.65	0.81
DecisionTree	RandomUndersampling 50/50	0.61	0.77
RandomForest	RandomUndersampling 50/50	0.66	0.82
LogisticRegression	RandomUndersampling 64/36	0.65	0.84
KNN	RandomUndersampling 64/36	0.66	0.85
DecisionTree	RandomUndersampling 64/36	0.63	0.84
RandomForest	RandomUndersampling 64/36	0.68	0.86
LogisticRegression	RandomOverSampling 50/50	0.64	0.81
KNN	RandomOverSampling 50/50	0.63	0.80
DecisionTree	RandomOverSampling 50/50	0.61	0.78
RandomForest	RandomOverSampling 50/50	0.67	0.87
LogisticRegression	RandomOverSampling 60/40	0.64	0.85
KNN	RandomOverSampling 60/40	0.64	0.85
DecisionTree	RandomOverSampling 60/40	0.62	0.85
RandomForest	RandomOverSampling 60/40	0.65	0.87

Concernant l'interprétabilité, on peut regarder l'importance des features pour la régression logistique avec le RandomUnderSampling. Cette analyse montre que:

- Les informations liées au climat et à la date d'observation n'ont finalement que peu d'importance

- Les données physiques (pluviométrie, humidité, pression, vent...) ont par contre bien plus d'intérêt; y compris les données que nous avons calculé pour augmenter notre jeu de données



## c. Classification multi-classes

Une classification multi-classes a été étudiée au début du projet; avant de se concentrer sur d'autres modèles.

Les détails sont disponibles dans le notebook associé.

## d. Régression

Dans le cadre de la régression, le but est de déterminer la quantité de pluie (en mm) qu'il y aura le lendemain.

Cette étude sera faite sur 2 échelles de temps: quotidienne et mensuelle.

3 métriques principales seront observées sur les jeux d'entraînement et de test:

- R2 score
- MSE
- MAE

### i. Echelle quotidienne

De nombreux modèles linéaires et non-linéaires ont été testé. Au delà des paramètres de chaque modèle, les leviers suivants ont été envisagés:

- Transformation des données d'entrées: PolynomialFeatures, MinMaxScaler, PowerTransformer
- Transformation des données de sorties: log, 1/x, PowerTransformer
- Sélection d'un nombre réduit de features

Le jeu de test est composé de 20% du jeu de données total.

Les résultats sont exposés dans les tableaux ci-après.

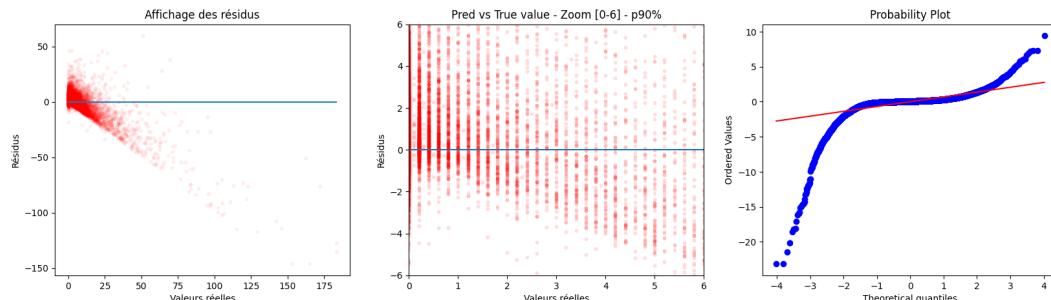
## LinearRegression

Les résultats sur la régression linéaire simple sont peu probants même s'ils peuvent être sensiblement améliorés en transformant les données d'entrées grâce à la fonction `PolynomialFeatures` de `sklearn.preprocessing`.

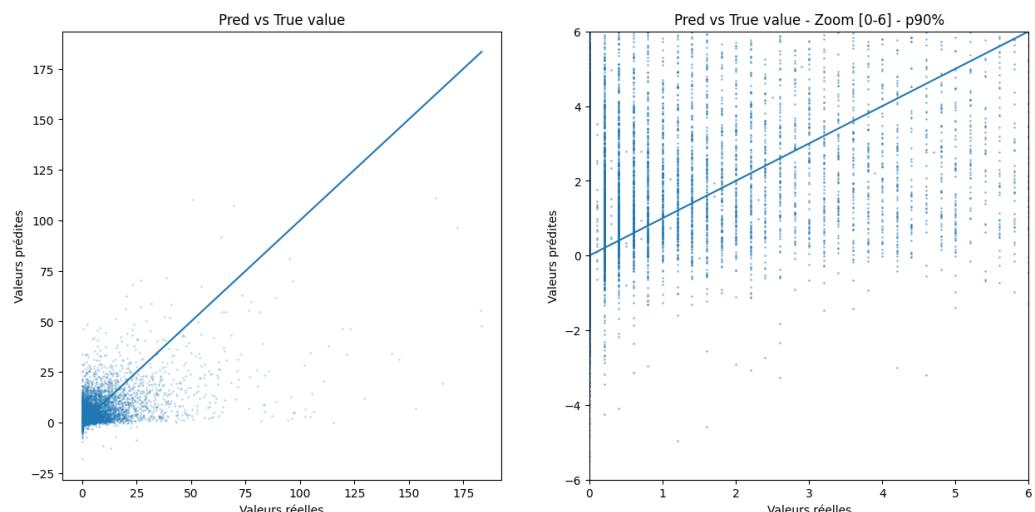
Une transformation de degré 3 a été retenue. Il n'a pas été possible d'augmenter le degré au-delà de 3 en raison de limitations matérielles mais il est probable que cette piste puisse permettre d'améliorer encore les résultats.

Features_number	X_transform	Notes	r2_train	r2_test	mse_train	mse_test	mae_train	mae_test
19	MinMaxScaler	-	0.22	0.22	57.08	47.09	3.23	3.13
5	MinMaxScaler	SelectFromModel	0.21	0.2	58.18	48.06	3.23	3.12
19	PowerTransformer(yeo-johnson)	-	0.18	0.18	60.5	49.34	3.38	3.27
364	PolynomialFeatures deg3	-	0.37	0.34	46.62	39.92	2.53	2.44
182	PolynomialFeatures deg3	50% top features	0.35	0.33	48.11	40.63	2.61	2.51

Répartition des résidus et Q-plot pour la régression linéaire + transformation Polynomiale de degré 3:



Répartition des prédictions pour la régression linéaire + transformation Polynomiale de degré 3:



### Autres modèles linéaires

Quelques autres modèles linéaires ont été testés. Ces modèles n'ont pas démontré de résultats plus intéressants que la régression linéaire simple malgré les temps de calculs plus longs.

Ces modèles n'auront donc pas été investigués plus en avant.

Model_name	Features_number	X_transform	notes	r2_train	r2_test	mse_train	mse_test	mae_train	mae_test
SVM	19	MinMaxScaler	-	0.22	0.22	57.09	47.04	3.22	3.12
Ridge	11	MinMaxScaler	-	0.22	0.21	57.56	47.52	3.24	3.13
SGDRegressor	11	MinMaxScaler	-	0.21	0.21	57.87	47.54	3.26	3.15
LinearSVR	11	-	-	0.2	0.19	59.12	49.24	3.65	3.55

### Modèles non-linéaires

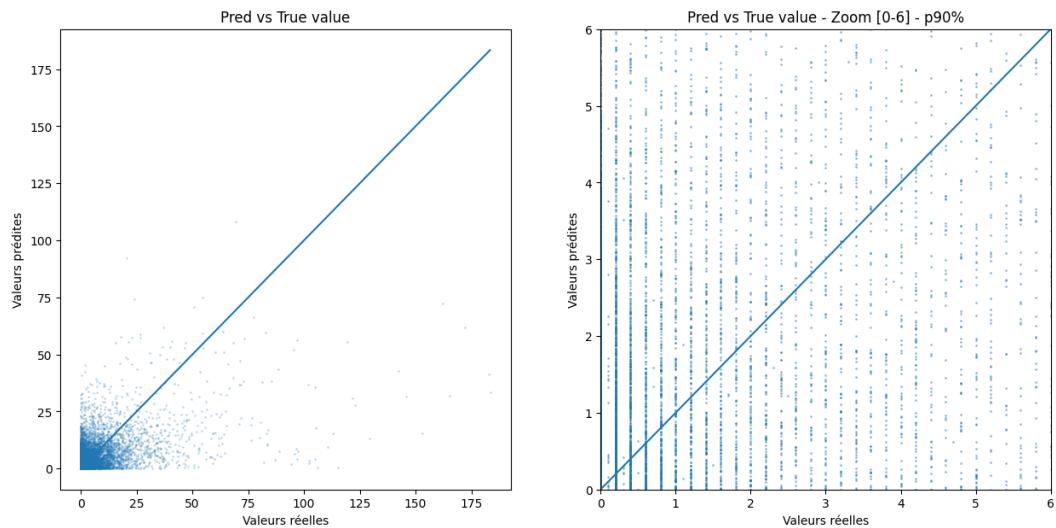
3 modèles non linéaires ont été testé. Un GridSearchCV a été utilisé dans un premier temps pour obtenir une valeur optimale pour chaque modèle.

Model_name	Features_number	X_transform	r2_train	r2_test	mse_train	mse_test	mae_train	mae_test
DecisionTreeRegressor	11	-	0.28	0.19	52.9	49.2	1.92	1.92
KNNRegressor	11	MinMaxScaler	1	0.32	0	41.2	0	2.23
RandomForestRegressor	11	-	0.5	0.31	36.5	41.6	2.29	2.36
HistGradientBoostingRegressor	364	PolynomialFeatures deg3	0.56	0.33	32.6	40.7	2.18	2.32

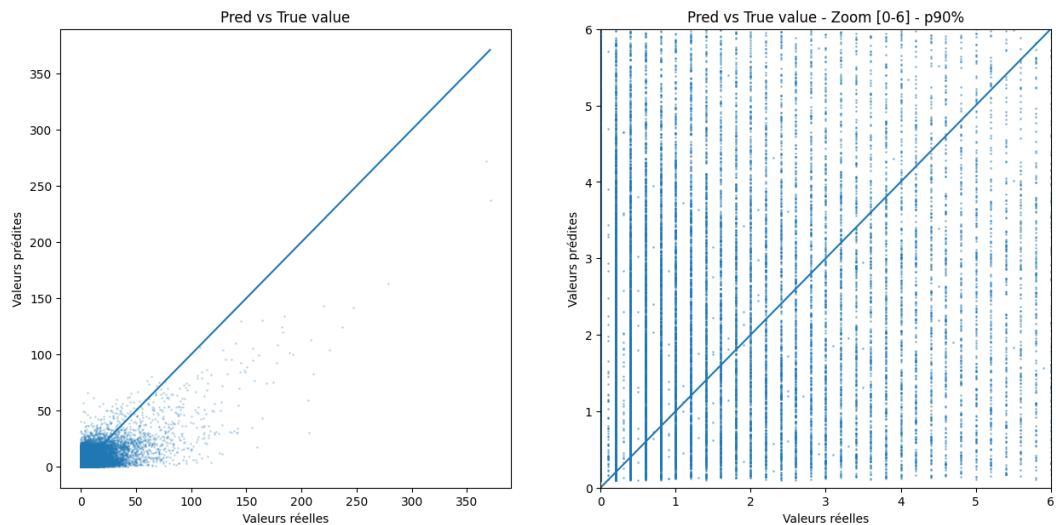
Ces modèles ont tous présenté du sur-entraînement.

Les modèles de type KNN, RandomForest et HistGradientBoostingRegressor semblent par contre sortir du lot avec des nuages de points plus centrés autour de la droite  $x=y$  (valeur prédite = valeur réelle):

### *Répartition des prédictions pour KNNRegressor:*



### *Répartition des prédictions pour RandomForest:*



#### **A noter:**

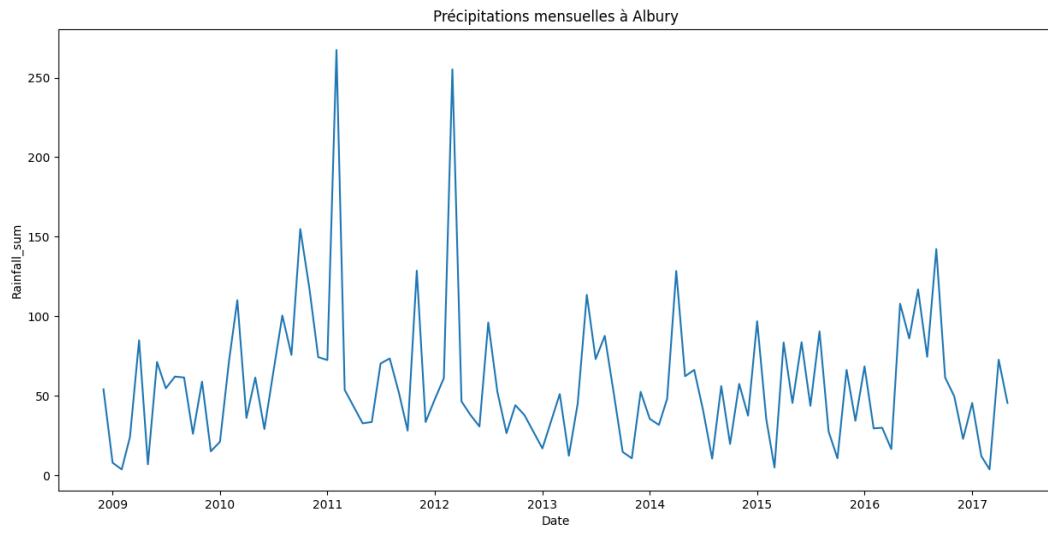
Le modèle HistGradientBoostingRegressor est particulièrement intéressant pour sa faible consommation en ressources matérielles.

#### **A investiguer:**

Avec plus de ressources matérielles, on peut penser que le couplage d'une transformation polynomiale de degré 3 avec un modèle non-linéaire fournira une précision accrue.

## ii. Echelle mensuelle

En ré-échantillonnant les données sur une base mensuelle (via des moyennes, min, max, std...), on obtient une valeur cible moins aléatoire comme le montre l'exemple ci-dessous pour la ville d'Albury.



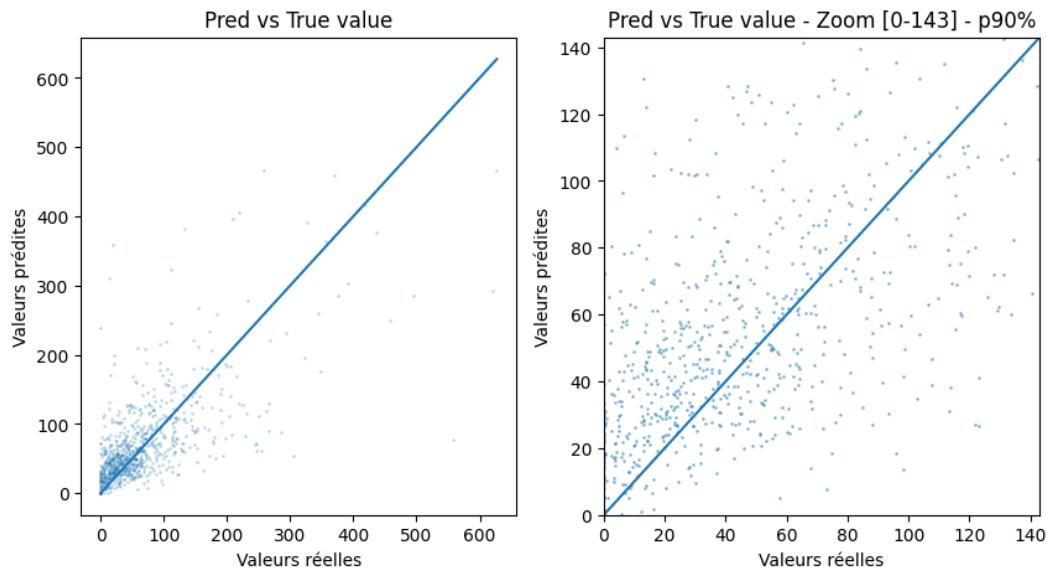
L'idée est donc d'observer la pertinence d'un modèle sur des données mensuelles. On cherchera à prédire les quantités de précipitations pour le mois suivant.

<b>Model_name</b>	<b>Features_number</b>	<b>X_transform</b>	<b>r2_train</b>	<b>r2_test</b>	<b>mse_train</b>	<b>mse_test</b>	<b>mae_train</b>	<b>mae_test</b>
LinearRegression	67	MinMaxScaler	0.34	0.27	4847	4056	44.2	43.0
KNNRegressor	67	MinMaxScaler	0.60	0.41	2930	3272	30.8	35.8
RandomForestRegressor	67	MinMaxScaler	0.92	0.36	601	3550	15.2	39.1

Les résultats restent très imprécis. Ils sont cependant sensiblement meilleurs que sur les prédictions quotidiennes.

On l'observe particulièrement en représentant les valeurs prédites en fonction des valeurs réelles. Le nuage de points est dispersé de manière relativement homogène autour de la diagonale.

### Exemple avec le modèle KNNRegressor:



### iii. Combinaison de modèles

Il est peut-être possible de combiner différents modèles afin de tirer parti de chacun d'entre eux et ainsi augmenter le résultat final.

A ce sujet, nous testerons différentes possibilités:

#### **StackingRegressor et VotingRegressor**

Ces 2 méthodes n'ont pas donné de résultat meilleurs qu'avec une simple régression linéaire.

#### **Moyenne de prédictions**

Il s'agit ici de combiner les prédictions de différents modèles de régression en en faisant une moyenne.

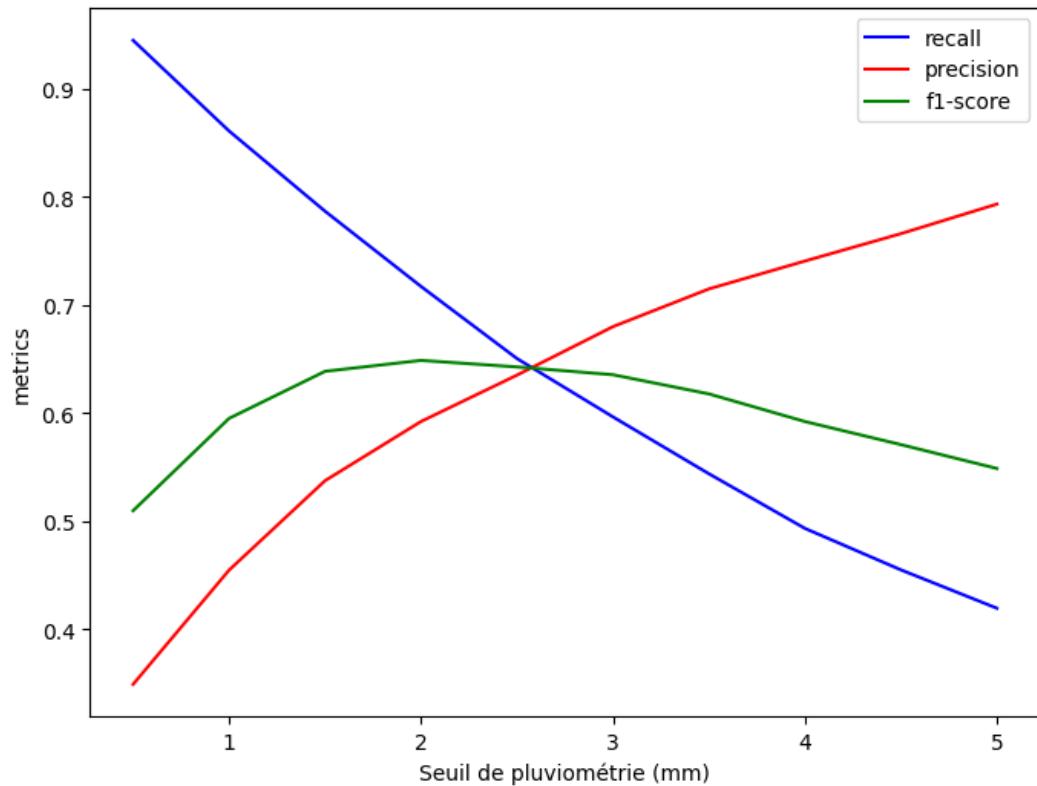
Les modèles sélectionnés sont les suivants en raison de leur diversité et de leurs résultats parmi les plus intéressants:

Model_name	Features_number	X_transform	r2_test
LinearRegression	364	PolynomialFeatures deg3	0.3358
HistGradientBoostingRegressor	364	PolynomialFeatures deg3	0.3348
KNNRegressor	11	-	0.2954

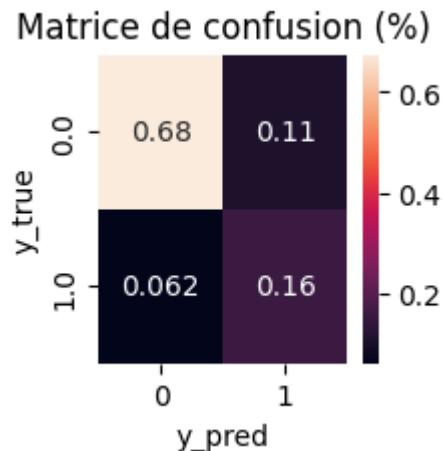
La moyenne des prédictions de chacun des modèles présentés ci-dessus permet d'obtenir un r2\_score de 0.3543; soit +5% par rapport au meilleur des 3 modèles.

On peut considérer qu'il y a un réel gain avec cette technique.

A partir de cette prédiction améliorée, on peut trouver un seuil afin d'obtenir une classification optimisée. Le graphique ci-dessous représente l'évolution des f1-score, recall et precision en fonction du seuil de pluviométrie choisi.



Si l'on choisit d'optimiser le f1-score, un seuil de 2mm semble pertinent pour définir une limite entre un jour pluvieux (1) ou non pluvieux (0).



Metrics:

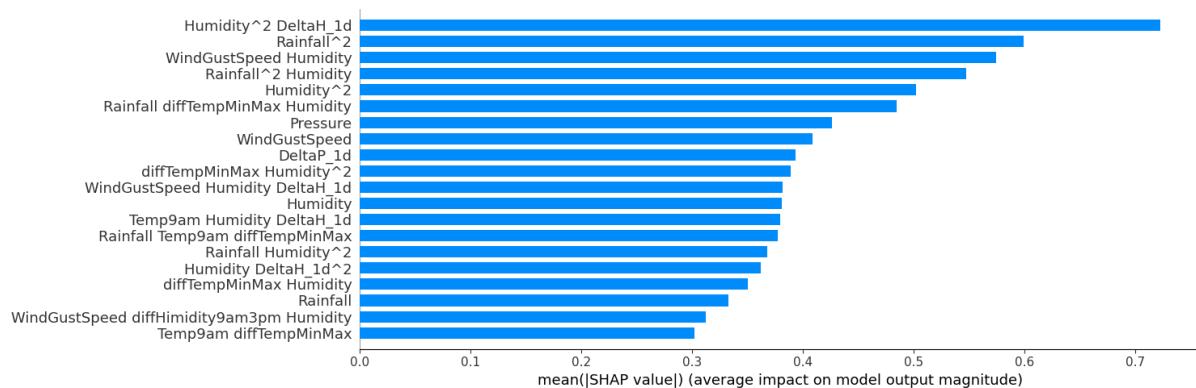
	Precision	Recall	F1 Score	Support
0	0.92	0.86	0.89	18649
1	0.59	0.72	0.65	5184

<b>Accuracy</b>			0.83	23833
<b>Macro avg</b>	0.75	0.79	0.77	23833
<b>Weighted avg</b>	0.85	0.83	0.84	23833

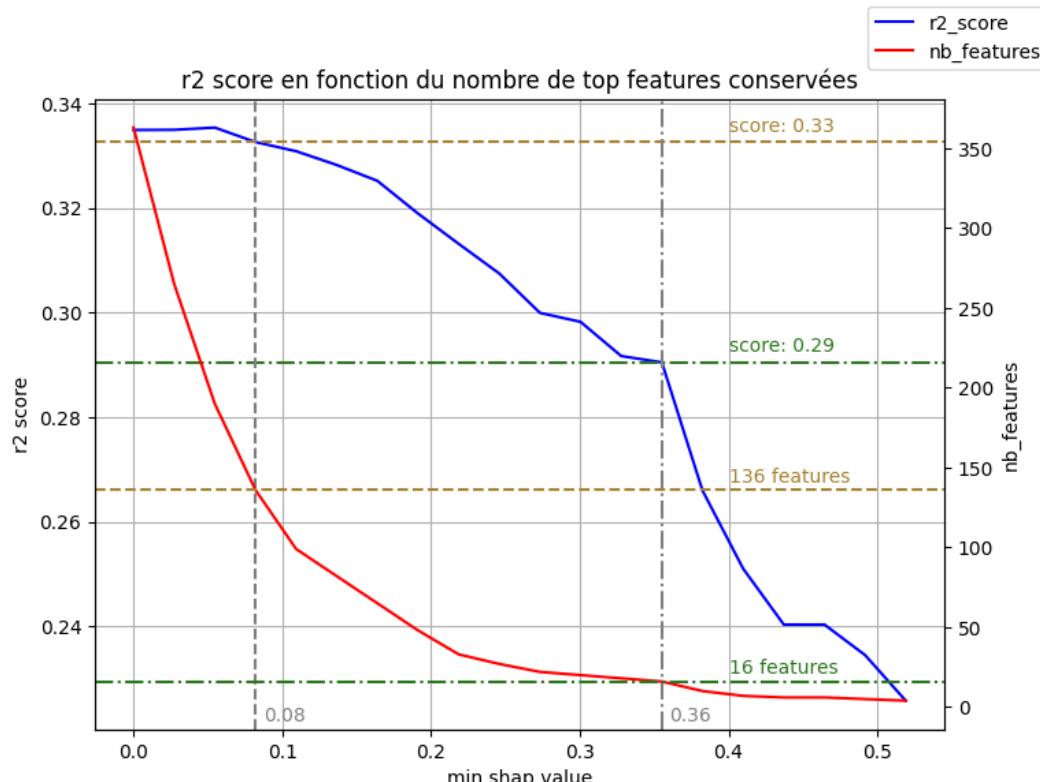
#### iv. Interprétabilité

Afin d'aller plus loin, on peut tenter d'interpréter les différentes features sur l'un des meilleurs modèles: LinearRegression + PolynomialFeatures(X, degré=3)

L'interprétabilité est cependant assez difficile au sens physique étant donné que l'on a affaire à des termes polynomiaux. Il est par contre aisément de définir les features les plus importantes:



Au delà de l'interprétation physique, il est intéressant de noter qu'en ne conservant que les features les plus importantes, on obtient un score avec relativement peu de perte sur le r2 score:



Le tableau suivant montre l'impact sur le r2\_score lorsque l'on se concentre sur les features polynomiales les plus importantes:

Model_name	Features_number	X_transform	r2_test
LinearRegression	364	PolynomialFeatures deg3	0.3349
LinearRegression	16 (-96%)	PolynomialFeatures deg3	0.2905 (-13%)
HistGradientBoostingRegressor	364	PolynomialFeatures deg3	0.3249
HistGradientBoostingRegressor	14 (-96%)	PolynomialFeatures deg3	0.3341 (+3%)
KNNRegressor	11	-	0.2954
KNNRegressor	16 (+45%)	PolynomialFeatures deg3	0.2839 (-4%)

On constate qu'on se concentrant sur seulement 14-16 features -soit environ 4% du nombre total de features- on n'impacte le r2 score que de façon assez limitée (<13%).

Par ailleurs, en combinant les modèles en moyennant les prédictions, on obtient un r2\_score de 0.32; soit un résultat tout à fait acceptable par rapport à toutes les régressions essayées.

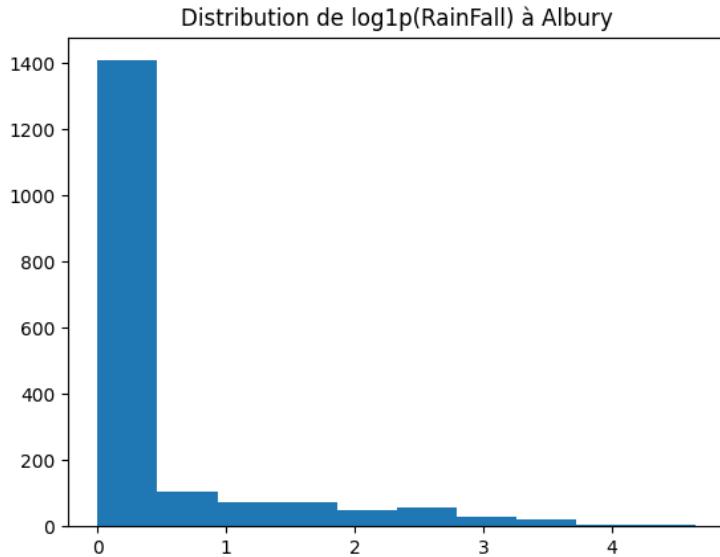
## e. Séries temporelles

### i. Introduction

On peut considérer les données météorologiques selon l'axe temporel et espérer qu'il existe une corrélation entre la météo du jour et celle des jours précédents.

On va utiliser des modèles de type ARIMA pour prévoir les précipitations dans le futur à partir des données du passé. Il y a cependant quelques limites imposées par ARIMA :

- La variable cible doit être une variable continue. On choisit donc de prédire le niveau de précipitations au lieu de prédire une variable booléenne indiquant la présence ou non de pluie. Compte tenu de la très forte dispersion de la distribution du niveau de précipitation, on applique une transformation en  $\log(1+x)$  de la valeur des précipitations pour rééquilibrer la distribution. Cette fonction a été choisie après le test de différentes méthodes de régularisation car elle donne les meilleurs résultats a posteriori.



- Le modèle ARIMA ne gère qu'un seul jeu de variables explicatives et une seule variable cible à une date donnée, ce qui signifie qu'un modèle ne peut prédire la météo que d'une seule ville. Il faudra entraîner autant de modèles que de villes.
- Dans le cas d'un modèle SARIMA, la période saisonnière doit être une valeur entière. Or nos données étant journalières, cela correspond à une périodicité de 365,25 jours. Afin d'obtenir une période entière, on enlève les jours correspondant à un 29 février pour obtenir une période unique de 365.

## ii. Préparation des données

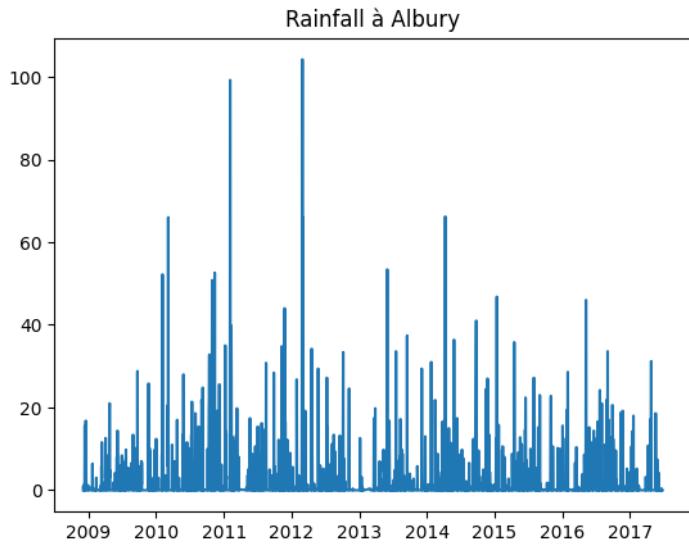
On applique les transformations suivantes aux données brutes :

- Restriction à une ville particulière
- Suppression des 29 février
- Réindexation avec un index temporel complet pour boucher les jours manquants
- Interpolation linéaire sur chaque colonne pour boucher les trous dans les données
- Standardisation des colonnes par StandardScaler
- Transformation par  $\log(1+x)$  de la variable cible
- Split temporel en jeux d'entraînement et de test : le jeu d'entraînement comprend les données entre le 01/07/2011 et le 30/06/2016, le jeu de test comprend les données à partir du 01/07/2016.

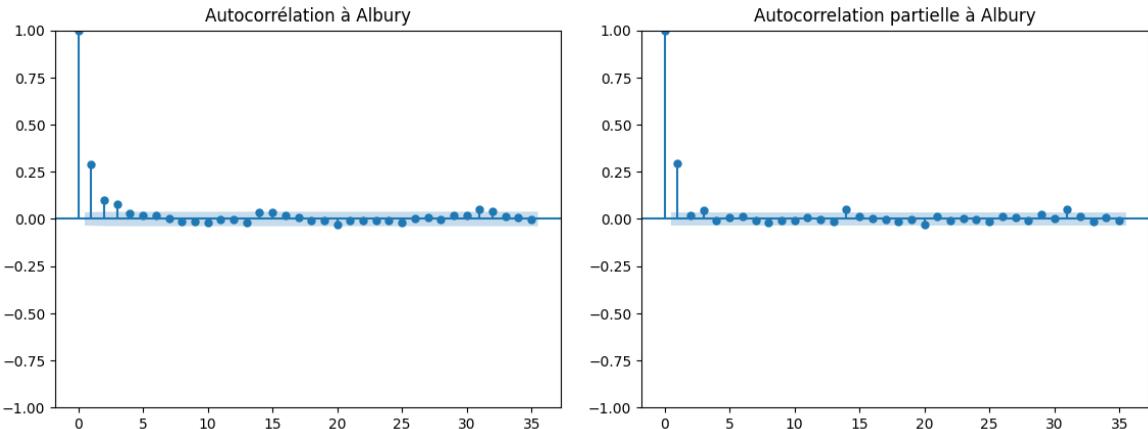
## iii. Estimation des paramètres ARIMA

Dans le cas d'un modèle ARIMA( $p, d, q$ ), il faut déterminer ses paramètres  $p$ ,  $d$  et  $q$ .

Bien que le rainfall soit une variable positive, elle vaut fréquemment zéro et on peut considérer qu'il n'y a pas de trend et donc qu'il n'y a pas de différentiation à appliquer ( $d=0$ ).



Pour déterminer  $p$  et  $q$ , on va analyser les diagrammes d'autocorrélation (ACF) et d'autocorrélation partielle (PACF).



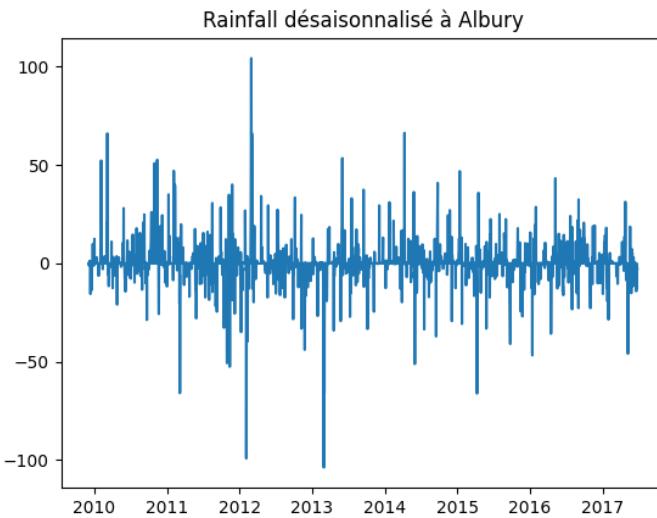
Le PACF présente une coupure nette à l'ordre 1 tandis que l'ACF présente une décroissance régulière. Ceci est la signature d'un modèle AR(1), c'est-à-dire  $p=1$  et  $q=0$ .

#### iv. Modèle SARIMAX

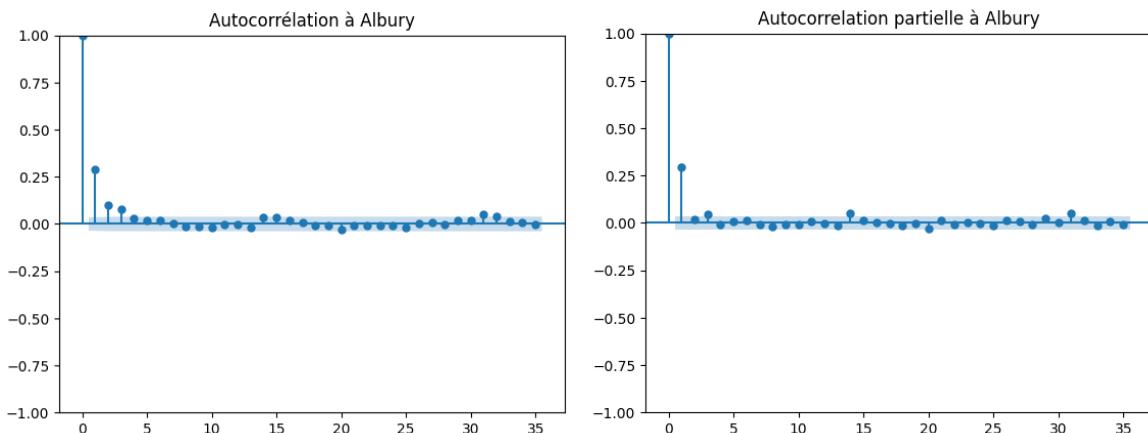
Il semble naturel de considérer que les phénomènes météorologiques possèdent une périodicité d'un an. Les données étant journalières, cela correspond à une périodicité de 365,25 jours. Les modèles ARIMA ne gèrent que des périodes entières, aussi on enlève des données les jours correspondant à un 29 février les années bissextiles pour obtenir une période de 365.

##### **Estimation des paramètres SARIMA**

Dans le cas d'un modèle SARIMA( $p, d, q$ )( $P, D, Q$ ), il faut déterminer ses paramètres  $P$ ,  $D$  et  $Q$ .



La visualisation de la différence Rainfall(T+365) - Rainfall(T) ne montre pas de trend. On choisit donc D=0.



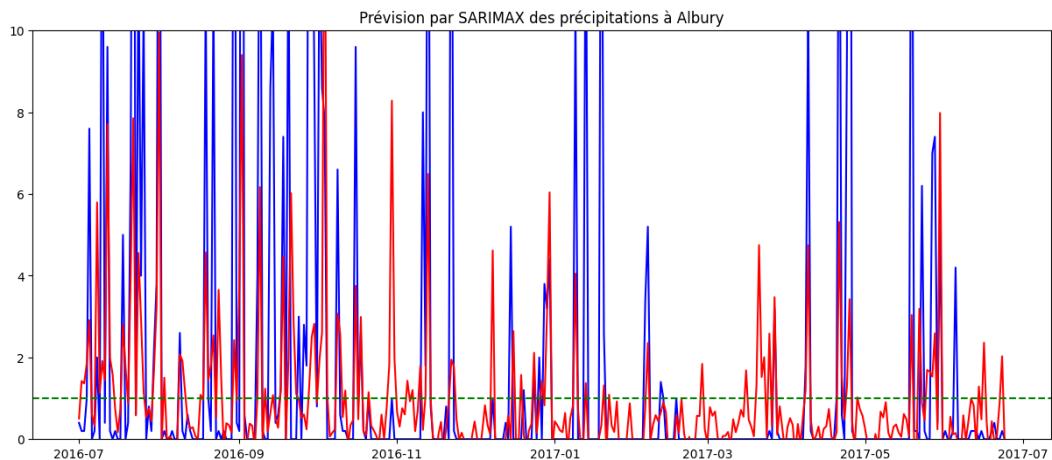
L'étude de l'ACF et du PACF montre de nouveau un modèle AR(1). Cependant, pour des raisons de performance de convergence et de qualité du résultat a posteriori, on choisit un modèle SARIMA(1, 0, 0)(0, 1, 1)365.

En effet, l'ajustement d'un modèle SARIMAX avec Statsmodels suppose une période courte (inférieure à 200). Statsmodels propose cependant un algorithme '*innovations\_mle*' permettant d'ajuster un modèle avec une longue période, mais avec des limitations sur les paramètres du modèle. De plus, l'ajustement prend plusieurs minutes.

## Résultats

L'estimation du modèle sur le jeu de test donne les résultats suivants :

## Prévision de rainfall

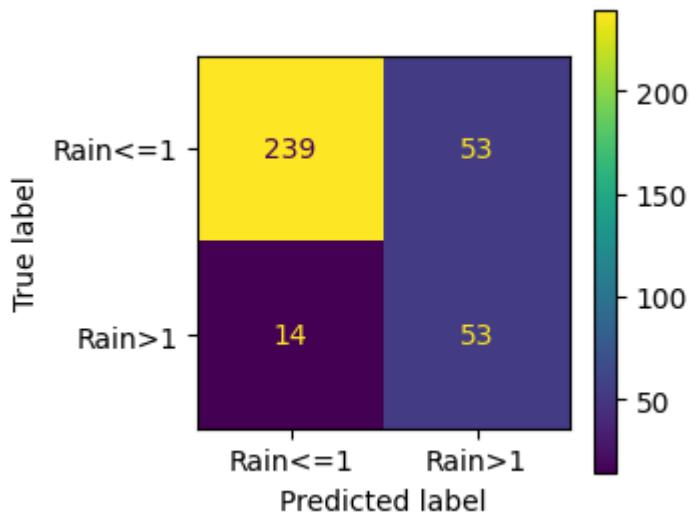


## Métriques

Erreur absolue moyenne (MAE) : 1.7404

	Precision	Recall	F1 Score	Support
<b>False</b>	0.94	0.82	0.88	292
<b>True</b>	0.50	0.79	0.61	67
<b>Accuracy</b>			0.81	359
<b>Macro avg</b>	0.72	0.80	0.74	359
<b>Weighted avg</b>	0.86	0.81	0.83	359

## Matrice de classification



## V. Modèle ARIMA

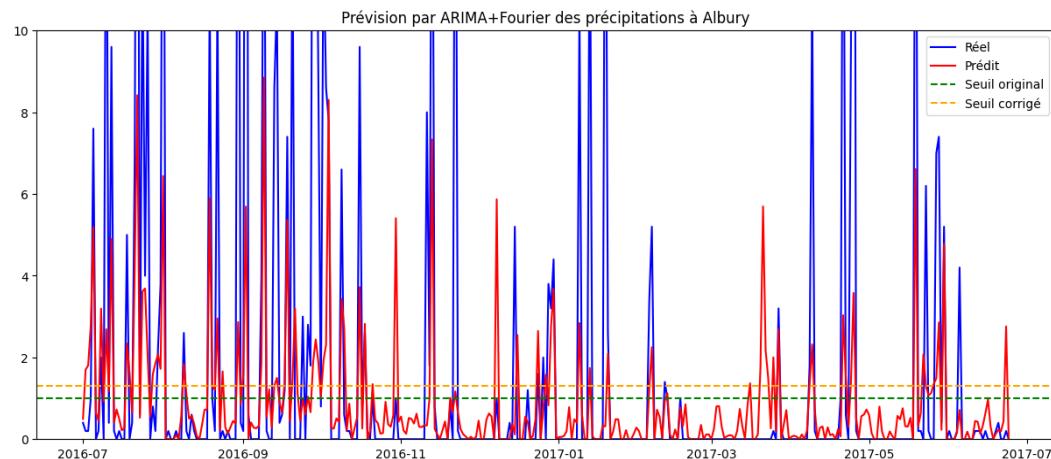
Comme proposé par Rob J. Hyndman sur le site <https://robjhyndman.com/hyndisght/longseasonality/>, on peut gérer les données avec une longue période saisonnière T au moyen d'un modèle ARIMA auquel on adjoint comme variables explicatives des features de Fourier  $\sin(k.t/T)$  et  $\cos(k.t/T)$  où k est le niveau d'harmonique.

En testant différentes valeurs du niveau d'harmoniques, un seul niveau donne des résultats satisfaisants.

### Résultats sur la ville d'Albury

L'estimation du modèle ARIMA sur le jeu de test donne les résultats suivants :

#### Prévision de rainfall

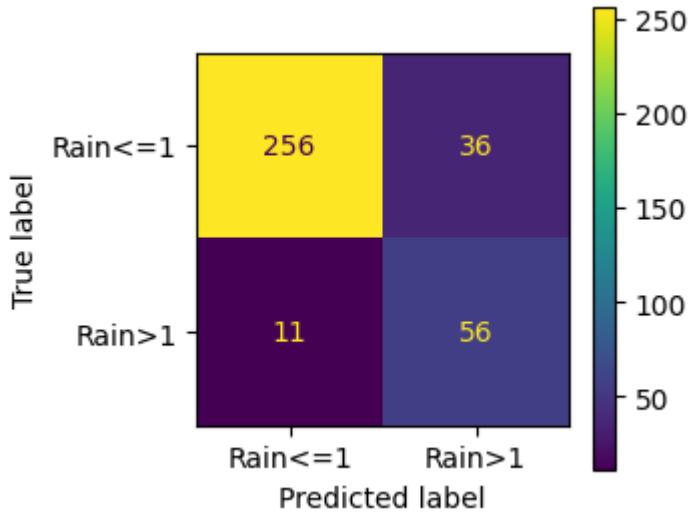


#### Métriques

Erreur absolue moyenne (MAE) : 1.555

	Precision	Recall	F1 Score	Support
False	0.96	0.88	0.92	292
True	0.61	0.84	0.70	67
Accuracy			0.87	359
Macro avg	0.78	0.86	0.81	359
Weighted avg	0.89	0.87	0.88	359

Matrice de classification



## vi. Ajustement du seuil

Le modèle ARIMA ne minimise pas l'erreur de classification mais l'écart quadratique entre les valeurs prédites et vraies de  $\log(1+Rainfall)$ , ce qui n'est pas la même chose. On peut supposer que cette transformation induit un écart sur la classification.

Pour essayer de compenser l'effet de cette transformation non linéaire, on peut chercher une fonction de transfert paramétrée pour transformer la valeur de Rainfall prédite, puis maximiser une métrique bien choisie par optimisation des paramètres de la fonction de transfert sur l'ensemble d'entraînement.

Cependant, comme au final on ne s'intéresse qu'au test  $Rainfall > 1$ , le problème peut se simplifier en remplaçant le test par  $Rainfall > \text{seuil}$  et en optimisant le paramètre seuil sur la métrique voulue. On choisit ici F1 Score(True) comme métrique à maximiser.

Afin d'éviter un “data leak”, ce calcul de la valeur de seuil optimale doit être effectuée sur l'ensemble d'entraînement et non sur l'ensemble de test.

### Résultats sur la ville d'Albury

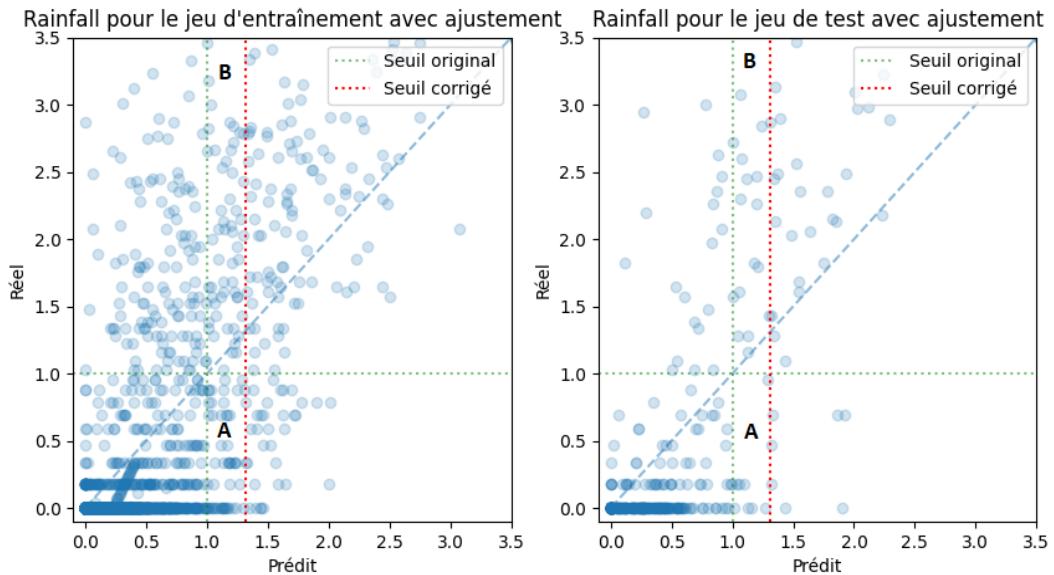
Seuil corrigé : 1.314

Classification avec ajustement du seuil

	Precision	Recall	F1 Score	Support
<b>False</b>	0.94	0.91	0.93	292
<b>True</b>	0.67	0.76	0.71	67
<b>Accuracy</b>			0.89	359
<b>Macro avg</b>	0.81	0.84	0.82	359

<b>Weighted avg</b>	0.89	0.89	0.89	359
---------------------	------	------	------	-----

### Visualisation de l'ajustement du seuil

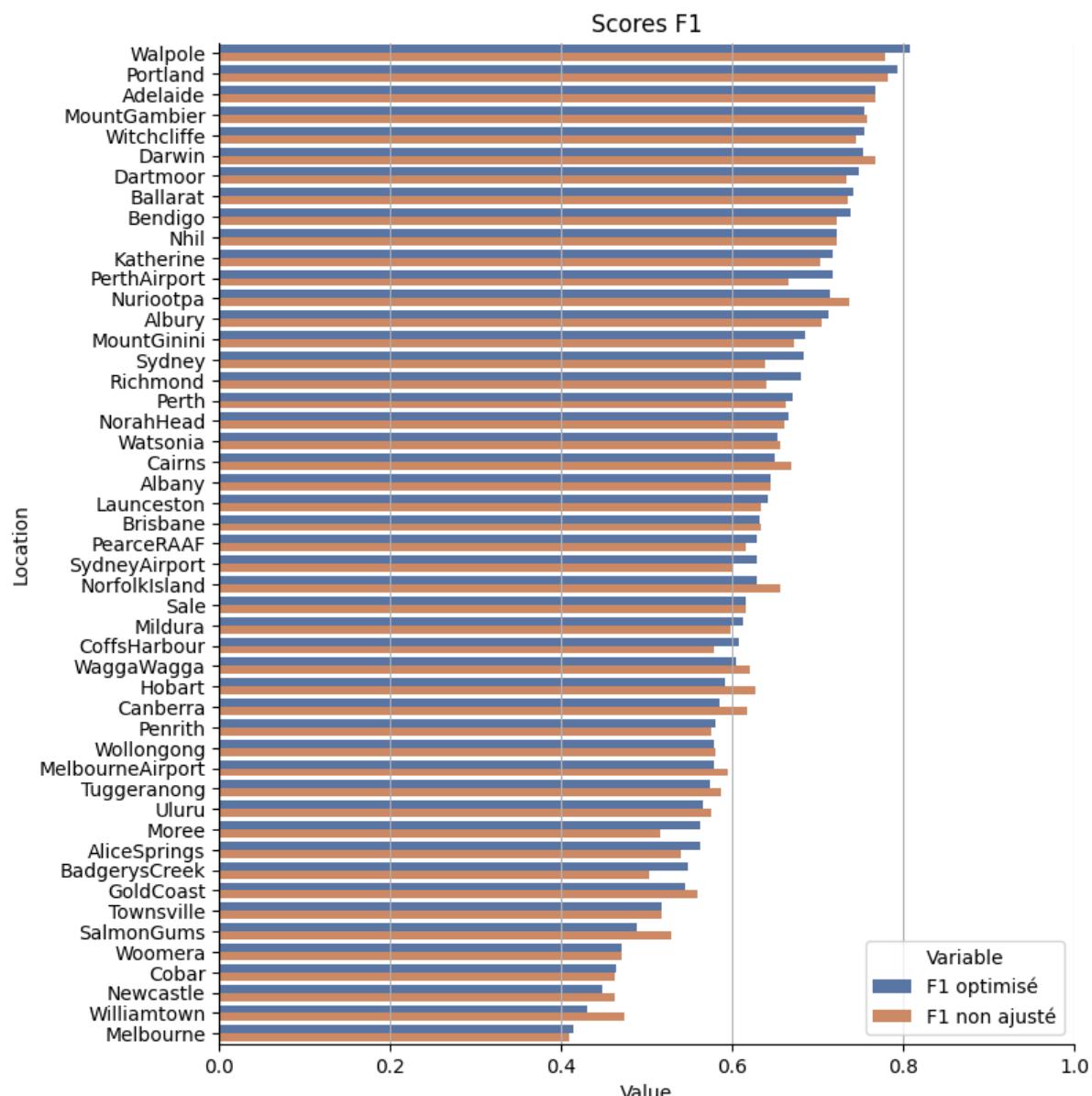


On voit que le fait de repousser le seuil à une valeur supérieure a un effet bénéfique lorsque les données sont plus concentrées dans la zone A que dans la zone B, car les données de la zone A se retrouvent intégrées dans les vrais négatifs tandis que celles de la zone B se retrouvent dans les faux négatifs.

En particulier, si la distribution des données dans le jeu de test est différente de ce point de vue de celle du jeu d'entraînement, l'ajustement du seuil peut conduire à une classification plus mauvaise que sans ajustement.

## vii. Calcul sur toutes les villes

L'ajustement des modèles pour chaque ville donne le résultat suivant :



On peut noter que le score F1 varie de manière importante selon les villes. De plus, on détecte bien un effet négatif de l'ajustement du seuil sur une fraction importante des villes.

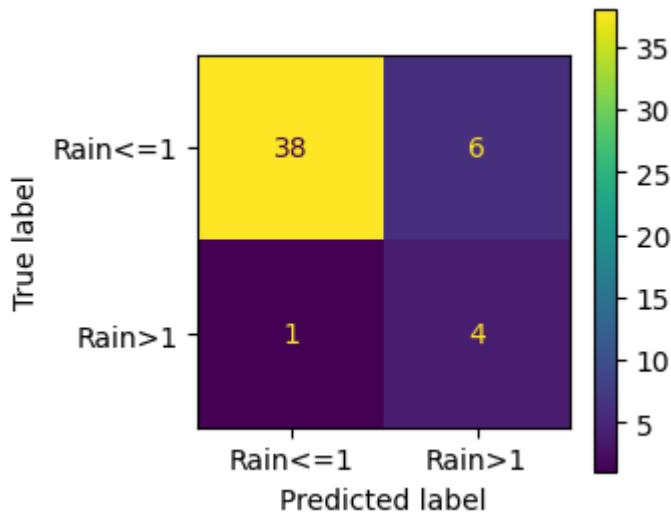
## viii. Détermination de la pluie pour le dernier jour

L'objectif du projet est de déterminer s'il pleuvra ou non le dernier jour de l'ensemble des données. On ré-entraîne alors les modèles de chaque ville sur l'ensemble des données et on les utilise pour prédire la pluviosité du dernier jour.

### Résultats sur l'ensemble des villes

#### Classification avec ajustement du seuil

	Precision	Recall	F1 Score	Support
<b>False</b>	0.97	0.86	0.92	44
<b>True</b>	0.40	0.80	0.53	5
<b>Accuracy</b>			0.86	49
<b>Macro avg</b>	0.69	0.83	0.72	49
<b>Weighted avg</b>	0.92	0.86	0.88	49

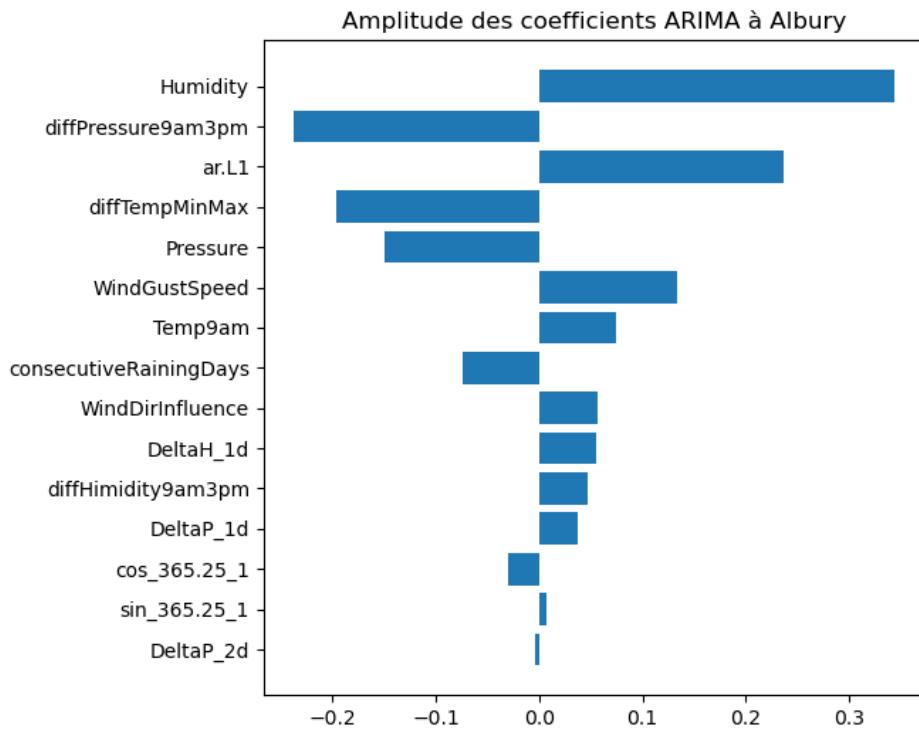


## ix. Interprétation du modèle

Le modèle ARIMA(1, 0, 0) a pour équation de prédiction :

$$\hat{Y}_t = \mu + \phi_1 Y_{t-1} + \sum_i \beta_i X_i$$

Il s'agit donc d'une équation linéaire facile d'interprétation puisque l'amplitude des coefficients  $\phi_1$  et  $\beta_i$  indique immédiatement l'importance de chaque terme.



On peut noter que le coefficient d'auto-régression  $\phi_1$  a une valeur importante (en troisième position) et donc que le terme auto-régressif a une importance majeure. La quantité de pluie d'un jour donné est fortement corrélée à celle du jour précédent.

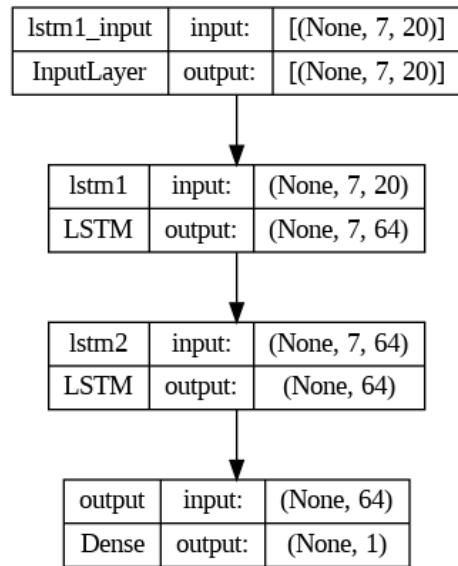
Hormis l'auto-régression, les paramètres les plus importants sont :

- un taux d'humidité élevé
- une chute de pression entre le matin et l'après-midi
- une faible hausse de température entre le matin et l'après-midi
- une basse pression

Inversement, on peut voir que les coefficients des features de Fourier ont une importance très faible et donc qu'il y a très peu d'influences saisonnières.

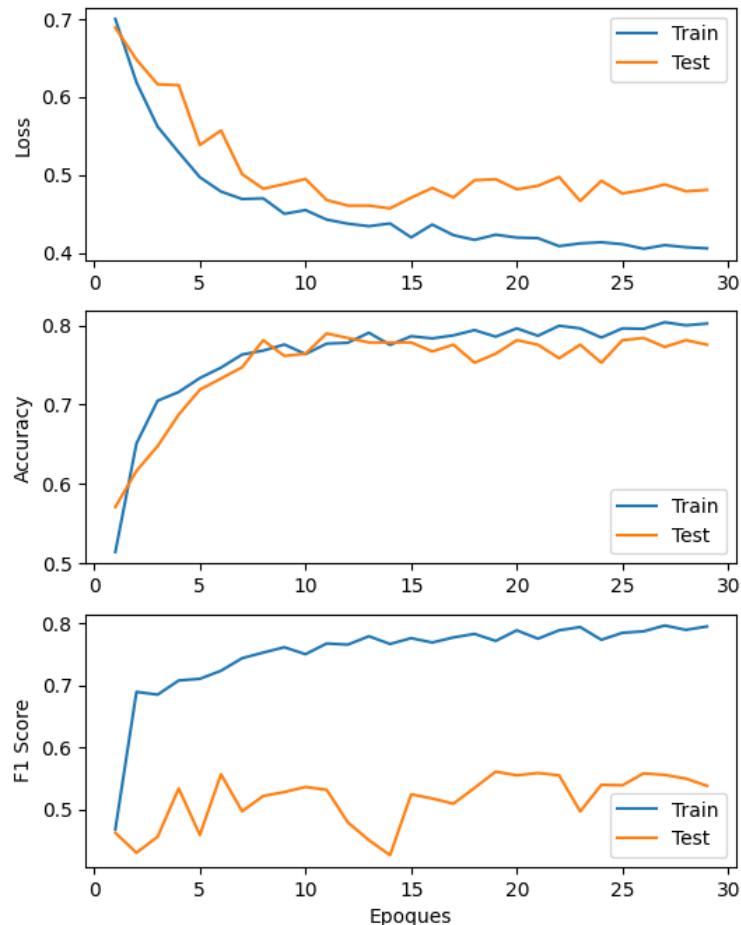
## f. Réseaux de neurones récurrents

On considère ici encore les données comme une série temporelle, mais on va les modéliser au moyen d'un réseau de neurones récurrents. On choisit une fenêtre de 4 jours au moyen l'architecture suivante :



La sortie du réseau est une fonction logistique indiquant s'il pleuvra ou non le lendemain.

L'entraînement du réseau converge vers une solution de qualité moyenne :



L'évaluation sur le jeu de test donne les résultats suivants :

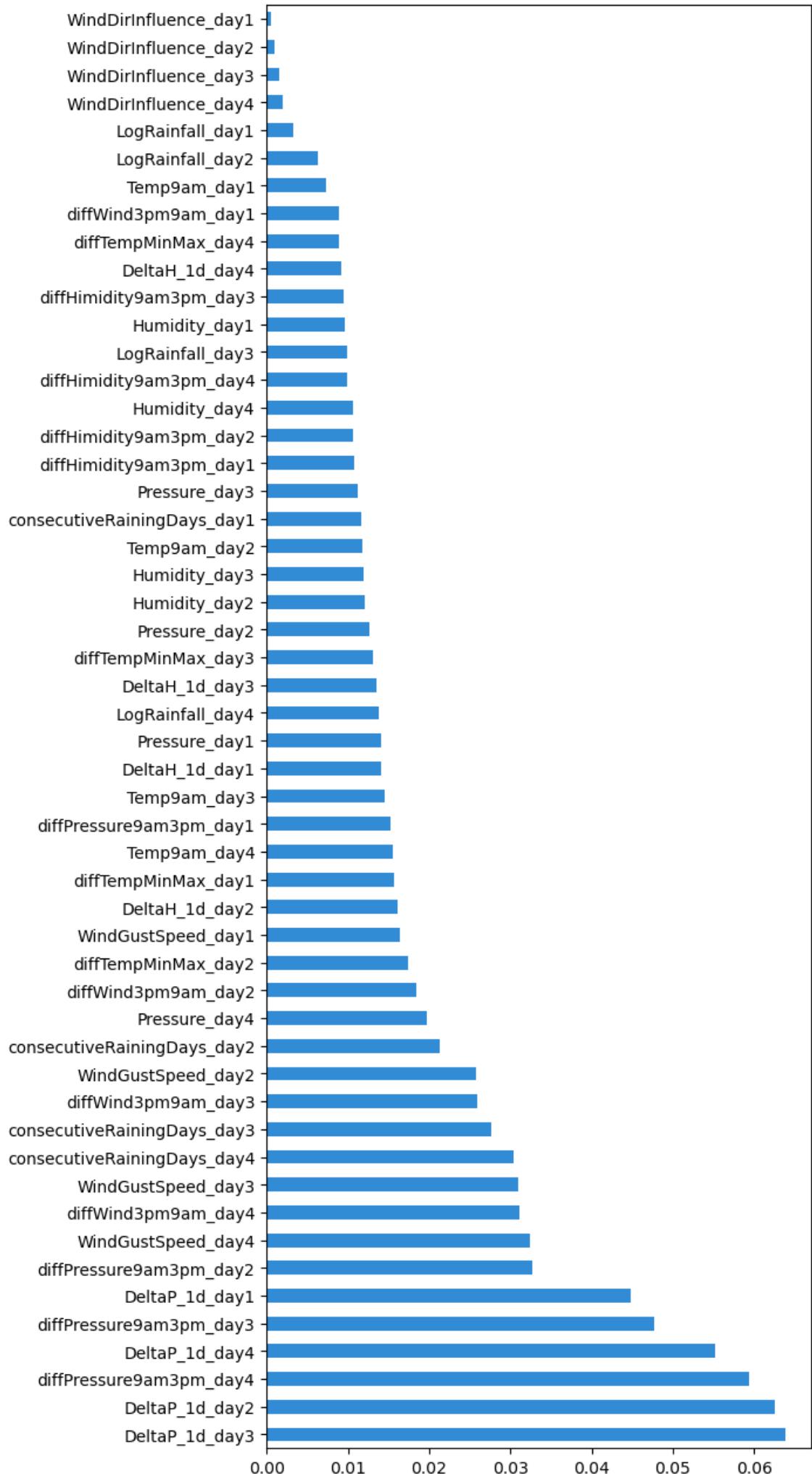
- loss: 0.4943
- accuracy: 0.7642
- precision: 0.5248
- recall: 0.6023
- f1\_score: 0.5608

### i. Interprétabilité du modèle RNN

#### **Interprétation globale avec Skater**

Skater permet de calculer l'influence globale des différents coefficients sur le résultat. Comme on s'appuie sur une fenêtre de 4 jours, les coefficients sont démultipliés sur les 4 jours de la fenêtre.

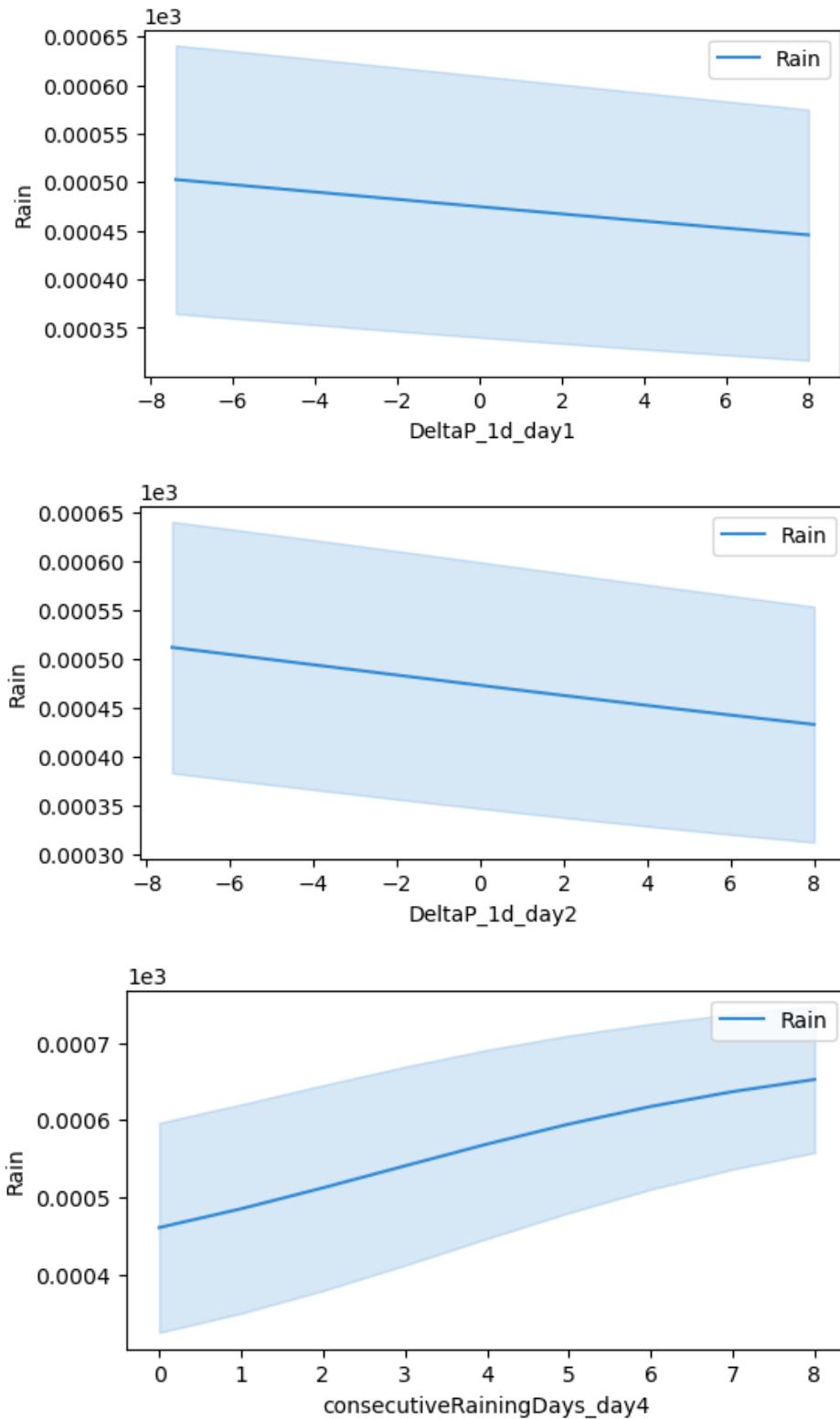
Le graphique ci-dessous indique l'influence de chaque paramètre où l'extension ‘\_dayn’ indique leur valeur  $n$  jours avant la date à estimer.



### Calcul de dépendance partielle avec Skater

Skater permet de calculer l'influence des paramètres sur le résultat comme une fonction de leur valeur, et non plus simplement sous forme d'un simple coefficient linéaire.

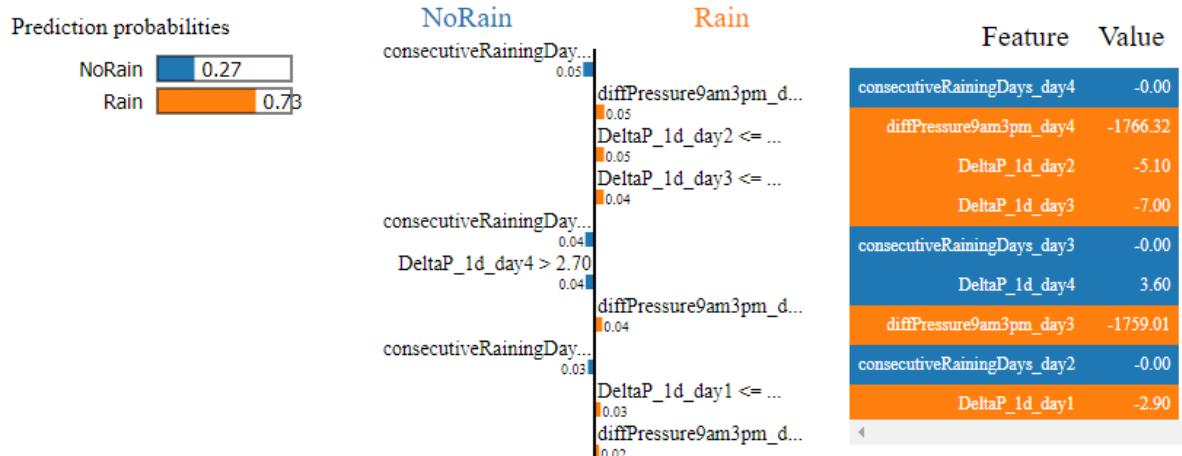
Le résultat sur quelques paramètres influents est visualisé ci-dessous :



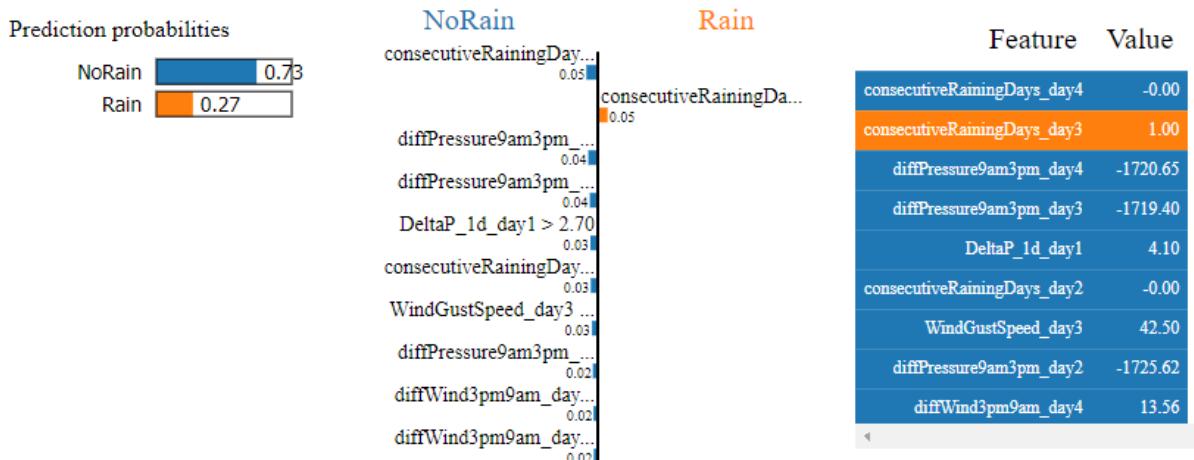
### Interprétation locale avec LIME

Le module LIME permet de calculer l'influence des différents paramètres pour une date donnée.

Par exemple, pour le jour 14 à Melbourne avec une probabilité forte de pluie :



et le jour 100 avec une probabilité faible de pluie :



## 5. Bilan et suite du projet

Le tableau suivant résume l'ordre de grandeur des meilleurs résultats que nous avons obtenus:

Modèle	Metric (meilleurs scores)
Classification KNeighborsClassifier sans rééchantillonage	f1-score=0.66
Régression	r2-score=0.35 (f1-score=0.65)
Série temporelle	f1-score=0.61 (jusqu'à 0.8 selon la ville)
Réseau de neurones	f1-score=0.56

Il est intéressant de noter qu'avec quelques optimisations, nous avons chacun réussi à obtenir des scores similaires malgré des méthodes de simulation très différentes.

Nos résultats montrent qu'avec du machine learning, nous réussissons à améliorer la prédiction par rapport à des modèles naïfs. Cependant, il semble malgré tout extrêmement compliqué de prédire avec précision la pluie à J+1 avec le jeu de données dont on dispose.

Si GrouseLager souhaite utiliser nos modèles de prédiction, l'entreprise devra le faire en gardant à l'esprit que l'indice de confiance est faible. Il doit s'agir là d'un outil utilisé en complément d'autres processus de décision. Cet outil ne se suffit pas à lui-même pour prédire la pluie.

Nos résultats sont malgré tout encourageants pour 2 raisons principales:

- Nous avons réalisé ces études avec très peu de ressources
- Les modélisations ont déjà permis -pour certaines villes- d'apporter des prédictions que nous jugeons utilisables.

Pour aller plus loin, nous recommandons de financer:

- l'acquisition de bases de données météorologiques plus fournies en données exploitables, car cette base est notoirement lacunaire sur certains types de données et les données présentes y sont extrêmement basiques.
- des ressources matérielles conséquentes afin de pouvoir entraîner des modèles très complexes

Exemples de besoins permettant d'améliorer nos prédictions :

- Disposer d'un maillage bien plus conséquent de stations météorologiques. En effet, pour couvrir toutes l'Australie, nos données ne proposent que 49 sites. De plus, ces sites sont très mal répartis sur le territoire.
- Disposer d'un relevé toutes les 3 heures, au lieu d'un relevé par 24h avec quelques informations portant sur des phénomènes météorologiques relevés dans la matinée (9h) et dans l'après-midi (15h). Ces informations augmenteraient le nombre de prévisions disponibles avec un pas de quelques heures au lieu d'une absence d'informations pendant 18h (de 15h à 9h le lendemain matin).

- Nos données n'exploitent pas la troisième dimension. Il serait intéressant d'y ajouter des données topographiques, relevé/observation multi-couche des colonnes d'air (utilisées par les modèles traditionnels de prévision).
- Nous avons volontairement supprimé quelques variables en raison du trop grand nombre de valeurs manquantes. Pourtant, ces variables semblaient assez bien corrélées à la variable cible; en particuliers la variable "nébulosité totale". De plus, la connaissance du type de nuages et de leur altitude estimée pourrait nous apporter de précieuses informations manquantes.

Nous pourrions acquérir différents types de relevés sur la couverture nuageuse, vus du sol mais avec un découpage à minima en 3 couches, avec par exemple :

- nébulosité des nuages de l'étage inférieur,
- hauteur de la base des nuages de l'étage inférieur,
- nébulosité de la couche inférieure,
- type de nuages de l'étage inférieur,
- type de nuages de l'étage moyen,
- type de nuages de l'étage supérieur,
- nébulosité totale

(référence

<https://www.data.gouv.fr/fr/datasets/observation-meteorologique-historiques-france-synop/>

# Annexes - Notebooks

Les notebooks respectifs sont disponibles sur le GitHub de notre projet:

<https://github.com/ArnoMac/weatherAUS/tree/main/notebooks>

Ci-dessous les noms de fichier de chaque notebook.

## Exploration et pre-processing

<i>Exploration des données</i>	<i>KaPy - Exploration des données.ipynb</i>
<i>Pre-processing &amp; Features engineering</i>	<i>KaPy - Step2- v3 - Pre-processing et features engineering.ipynb</i>

## Modélisation et interprétabilité

<i>Classification binaire sans rééchantillonnage</i>	<i>voir KNeighborsClassifier-Annexes.zip :</i> <ul style="list-style-type: none"><li>• <i>Indicateurs en sortie de simulation - indispensable pour le reporting bokeh.zip</i></li><li>• <i>KaPy - Reéquilibrage des données - ClusterCentroids - KNN local.ipynb</i></li><li>• <i>KNN avec modif des données ponderation des variables multiplication par information mutuelle.ipynb</i></li><li>• <i>KNN Mutual Information - fonction distance_mi - puis simulation avec pondération du dataset.ipynb</i></li><li>• <i>KNN Simulation avec la fonction de pondérations weights_sqr.ipynb</i></li><li>• <i>KNN Simulations en faisant varier les principaux paramètres.ipynb</i></li><li>• <i>KNN Simulations PCA avec reduction de dimensions.ipynb</i></li><li>• <i>KNN Simulations PCA sans reduc de dim.ipynb</i></li></ul>
<i>Classification binaire avec rééchantillonnage</i>	<i>Classification Binaire avec Rééchantillonage.ipynb</i>
<i>Classification à 3 classes</i>	<i>KaPy - Classification à 3 classes.ipynb</i>
<i>Régressions</i>	<i>KaPy - Régressions.ipynb</i>
<i>Régressions (interprétabilité et optimisations)</i>	<i>KaPy - Régression - Interprétabilité &amp; Optimisation.ipynb</i>
<i>ARIMA</i>	<i>KaPy - Précipitations par ARIMA.ipynb</i>
<i>LSTM (Réseau de neurones Récursifs)</i>	<i>KaPy - Précipitations par LSTM.ipynb</i>