

Ministère de l'Enseignement Supérieur et de la Recherche
Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique
Département IA et SD



Rapport Pour La Partie 02 De Projet

Module : Data Mining

Apprentissage supervisé et non supervisé

Travail présenté par :

- BENKOUTEN Aymen———-191931046409
- NOUGHI Tarek———-191931041952

2023/2024

Table des matières

I	Analyse et prétraitement des données	2
1	Objectifs	2
2	Manipulation de dataset	2
3	Analyse des caractéristiques des attributs du dataset	4
3.1	Calcul des mesures de tendance centrale pour déduire les symétries . . .	4
3.2	Construction des boîtes à moustache pour la détections des données aberrante	7
3.3	Construction des histogrammes des données	9
3.4	Construction des diagrammes de dispersion des données	10
4	Prétraitement	12
4.1	Traitement des valeurs manquantes et aberrantes	12
4.1.1	Choix de la méthode de remplacement des valeurs manquantes .	12
4.1.2	Choix de la méthode de traitement des valeurs aberrantes . . .	13
4.2	Réduction des données (élimination des redondances)	18
4.2.1	verticales	18
4.2.2	horizontales	18
4.3	Normalisation des données	18
4.3.1	Méthode Min-Max	19
4.3.2	Méthode z-score	19
II	Analyse supervisée	20
1	Objectif	20
2	Méthodologie :	20
2.1	Préparation des Données :	20
2.2	Sélection des Modèles de Classification :	20
2.3	Application des Modèles :	21
2.3.1	Méthodes d'application des trois algorithmes sur les instances du dataset :	21
2.3.2	Illustration par des Exemples de l'Application des Modèles . . .	23
3	Évaluation des Modèles	24
3.1	Explication sur la manière de donner la matrice de confusion	24
3.2	Méthodologie d'Évaluation des Modèles de Classification	26
3.3	Analyse et Comparaison des Performances des Modèles de Classification	28
3.4	Évaluation et Comparaison Globale	28

III	Analyse non supervisée	29
1	Objectif	29
2	Méthodologie :	29
2.1	Préparation des Données :	29
2.2	Sélection des Modèles de Clustering :	29
2.3	Application des Modèles :	29
2.4	Expérimentation en variant les paramètres de k-means et DBSCAN . . .	31
3	Comparaison des Algorithmes	38
IV	Annexe	41
4	Dépendance du code :	42

Table des figures

1	Graphe Densité de N, K, E, C, S, P, Pc, Oc et Zn	5
2	Graphe Densité de MN, B, OM, Fertility,Fe et Cu	6
3	Boîte à moustache	7
4	Boîte à moustache (suite)	8
5	Histogrammes	9
6	Histogrammes (suite)	10
7	La matrice de corrélation de dataset-1-	11
8	La matrice de corrélation de dataset-1-	12
9	Les valeurs nulles pour chaque attributs	13
10	Boite a moustache sans valeurs aberrantes (moyenne)	14
11	Boite a moustache sans valeurs aberrantes (suite - moyenne)	15
12	Boite a moustache sans valeurs aberrantes (mediane)	16
13	Boite a moustache sans valeurs aberrantes (suite - mediane)	17
14	Élimination des valeurs aberrantes de B	18
15	Portion Résultat Normalisation Min-Max	19
16	Portion Résultat normalisées Z-score	19
17	Enter Caption	23
18	Fonction de Matrice de Confusion	24
19	Application de la Fonction	25
20	Les fonctions d'évaluation	26
21	Application de l'Évaluation	27
22	Heatmap pour les Métriques par Itération	32
23	Heatmap pour les Métriques par convergence	32
24	courde de coude	33
25	la répartition spatiale des clusters	34
26	Histogramme des tailles des clusters	34
27	Effet de la configuration DBSCAN sur le score de silhouette en fonction d'eps.	35
28	Influence du paramètre min_samples sur le score de silhouette.	36
29	Graphique en surface 3D montrant l'impact de l'interaction entre eps et min_samples sur le score de silhouette.	37
30	Évaluation du score de silhouette et du nombre de clusters pour DBSCAN	37
31	Visualisation DBSCAN Clustering (PCA).	38
32	Graphiques en barres des métriques de clustering.	39

Liste des tableaux

1	Description de dataset-1-	2
2	Statistiques des données	3
3	Statistiques des données (Suite)	3
4	Description des colonnes du dataset	3
5	Description statistique des attributs.	4
6	Comparaison des performances des modèles de classification	27
7	3 premiers lignes des Résultats de l'Exploration des Hyperparamètres	32

Introduction Générale

Les données, par leur volume et leur complexité croissants, constituent un enjeu stratégique majeur dans divers secteurs d'activité. En agriculture, la capacité à analyser et interpréter ces données peut conduire à des avancées significatives en matière de rendement et de gestion durable des ressources. Dans ce rapport, nous abordons le traitement et l'analyse d'un ensemble de données agricoles, en mettant un accent particulier sur les méthodes d'apprentissage supervisé et non supervisé.

La première partie de notre étude se consacre au clustering, une méthode d'analyse non supervisée permettant de découvrir des groupements naturels au sein d'un jeu de données sans étiquettes prédéfinies. Nous avons appliqué des algorithmes comme K-means et DBSCAN, en évaluant leurs performances à travers diverses métriques et visualisations. Cette approche nous aide à comprendre les structures cachées et à identifier les facteurs influençant la fertilité des sols.

Dans la seconde partie, nous nous concentrons sur l'analyse supervisée. Elle consiste à utiliser des données étiquetées pour entraîner des modèles capables de prédire ces étiquettes. Les algorithmes de classification tels que KNN et Decision Trees ont été programmés et testés. Nous avons cherché à prédire la fertilité des sols, un paramètre crucial pour la productivité agricole.

Première partie

Analyse et prétraitement des données

1 Objectifs

L'objectif principal de cette section consiste à conduire une analyse approfondie et à mettre en œuvre un processus de nettoyage du dataset 1. Cette phase revêt une importance cruciale, car elle vise à préparer ces données en vue de leur utilisation ultérieure pour la classification et le clustering.

Le dataset 1 renferme des informations relatives aux caractéristiques du sol, et sa préparation adéquate.

Les étapes clés de cette analyse incluront l'extraction d'informations cruciales, le nettoyage des données pour assurer leur fiabilité, et la création de visualisations significatives

2 Manipulation de dataset

Visualisation du contenu du dataset

	N	P	K	pH	EC	OC	S	Zn	Fe	Cu	Mn	B	OM	Fertility
0	138	8.6	560	7.46	0.62	0.70	5.90	0.24	0.31	0.77	8.71	0.11	1.2040	0
1	213	7.5	338	7.62	0.75	1.06	25.40	0.30	0.86	1.54	2.89	2.29	1.8232	0
2	163	9.6	718	7.59	0.51	1.11	14.30	0.30	0.86	1.57	2.70	2.03	1.9092	0
3	157	6.8	475	7.64	0.58	0.94	26.00	0.34	0.54	1.53	2.65	1.82	1.6168	0
4	270	9.9	444	7.63	0.40	0.86	11.80	0.25	0.76	1.69	2.43	2.26	1.4792	1
5	220	8.6	444	7.43	0.65	0.72	11.70	0.37	0.66	0.90	2.19	1.82	1.2384	0

TABLE 1 – Description de dataset-1-

Le dataset présenté dans le tableau 1.1 contient des informations liées à différents paramètres agronomiques et chimiques associés à des échantillons de sol. Les colonnes représentent les variables mesurées, telles que N (azote), P (phosphore), K (potassium), pH, EC (conductivité électrique), OC (carbone organique), S (soufre), Zn (zinc), Fe (fer), Cu (cuivre), Mn (manganèse), B (boron), OM (matière organique), et Fertility (fertilité).

Chaque ligne du dataset correspond à une observation ou un échantillon distinct, avec des valeurs spécifiques pour chaque variable. Par exemple, la première ligne indique des valeurs spécifiques pour N, P, K, pH, etc.

L'objectif de ce dataset pourrait être d'analyser la relation entre ces variables et la fertilité du sol, étant donné la présence de la colonne "Fertility". Une analyse de visualisation pourrait être réalisée pour mieux comprendre les tendances, les corrélations ou les schémas au sein des données c'est ce qu'on va voir par la suite.

Une description globale du dataset

	N	K	pH	EC	OC	S	Zn	Fe
Count	885.000	885.000	885.000	885.000	884.000	885.000	885.000	885.000
Mean	246.998	501.339	7.512	0.544	0.618	7.546	0.469	4.127
Std	77.359	129.105	0.465	0.141	0.841	4.418	1.889	3.108
Min	6.000	11.000	0.900	0.100	0.100	0.640	0.070	0.210
25%	201.000	412.000	7.350	0.430	0.380	4.700	0.280	2.050
50%	257.000	475.000	7.500	0.550	0.590	6.640	0.360	3.560
75%	307.000	581.000	7.630	0.640	0.780	8.750	0.470	6.320
Max	383.000	1560.000	11.150	0.950	24.000	31.000	42.000	44.000

TABLE 2 – Statistiques des données

	Cu	Mn	B	OM	Fertility
Count	884.000	885.000	885.000	885.000	885.000
Mean	0.952	8.654	0.593	1.064	0.592
Std	0.466	4.301	0.575	1.446	0.578
Min	0.090	0.110	0.060	0.172	0.000
25%	0.630	6.210	0.270	0.6536	0.000
50%	0.930	8.340	0.410	1.0148	1.000
75%	1.250	11.470	0.610	1.3416	1.000
Max	3.020	31.000	2.820	41.280	2.000

TABLE 3 – Statistiques des données (Suite)

Description de chaque attribut

Column	Non-Null-Count	Dtype	Unique-Value-Count
N	885 non-null	int64	61
P	885 non-null	object	93
K	885 non-null	int64	63
pH	885 non-null	float64	107
EC	885 non-null	float64	71
OC	884 non-null	float64	69
S	885 non-null	float64	153
Zn	885 non-null	float64	70
Fe	885 non-null	float64	387
Cu	884 non-null	float64	167
Mn	885 non-null	float64	429
B	885 non-null	float64	127
OM	885 non-null	float64	68
Fertility	885 non-null	int64	3

TABLE 4 – Description des colonnes du dataset

3 Analyse des caractéristiques des attributs du dataset

3.1 Calcul des mesures de tendance centrale pour déduire les symétries

	Moyenne	Médiane	Mode	Max	Min	q0	q1	q2	q3	q4
N	247.0	257	207	383	6	6	201.0	257	307.0	307.0
P	14.56	7.5	8.3	125.0	2.9	2.9	12.4	7.5	10.6	10.7
K	501.34	475	444	1560	11	11	412.0	475	581.0	581.0
pH	7.51	7.5	7.5	11.15	0.9	0.9	7.35	7.5	7.63	7.63
EC	0.54	0.55	0.62	0.95	0.1	0.1	0.43	0.55	0.64	0.64
OC	0.62	0.68	0.88	24.0	0.1	0.1	0.39	0.68	1.07	1.07
S	7.55	6.64	5.13	31.0	0.64	0.64	4.7	6.64	8.75	8.75
Zn	0.47	0.36	0.28	42.0	0.07	0.07	0.28	0.36	0.47	0.47
Fe	4.13	3.56	6.32	44.0	0.21	0.21	2.035	3.56	6.31	6.32
Cu	0.95	0.93	1.25	3.02	0.09	0.09	0.63	0.93	1.25	1.25
Mn	8.65	8.34	7.54	31.0	0.11	0.11	6.21	8.34	11.45	11.48
B	0.59	0.41	0.34	2.82	0.06	0.06	0.27	0.41	0.61	0.61
OM	1.06	1.01	1.51	41.28	0.172	0.172	0.6536	1.0148	1.34	1.34
Fertility	0.59	1	1	2	0	0	0.0	1	1.0	1.0

TABLE 5 – Description statistique des attributs.

Les symétriques sont les colonnes où la moyenne = la médiane = le mode, pour conclure ça visuellement nous exploitons les graphes de densité pour chaque attribut et on tire les symétriques on aura donc 14 graphes.

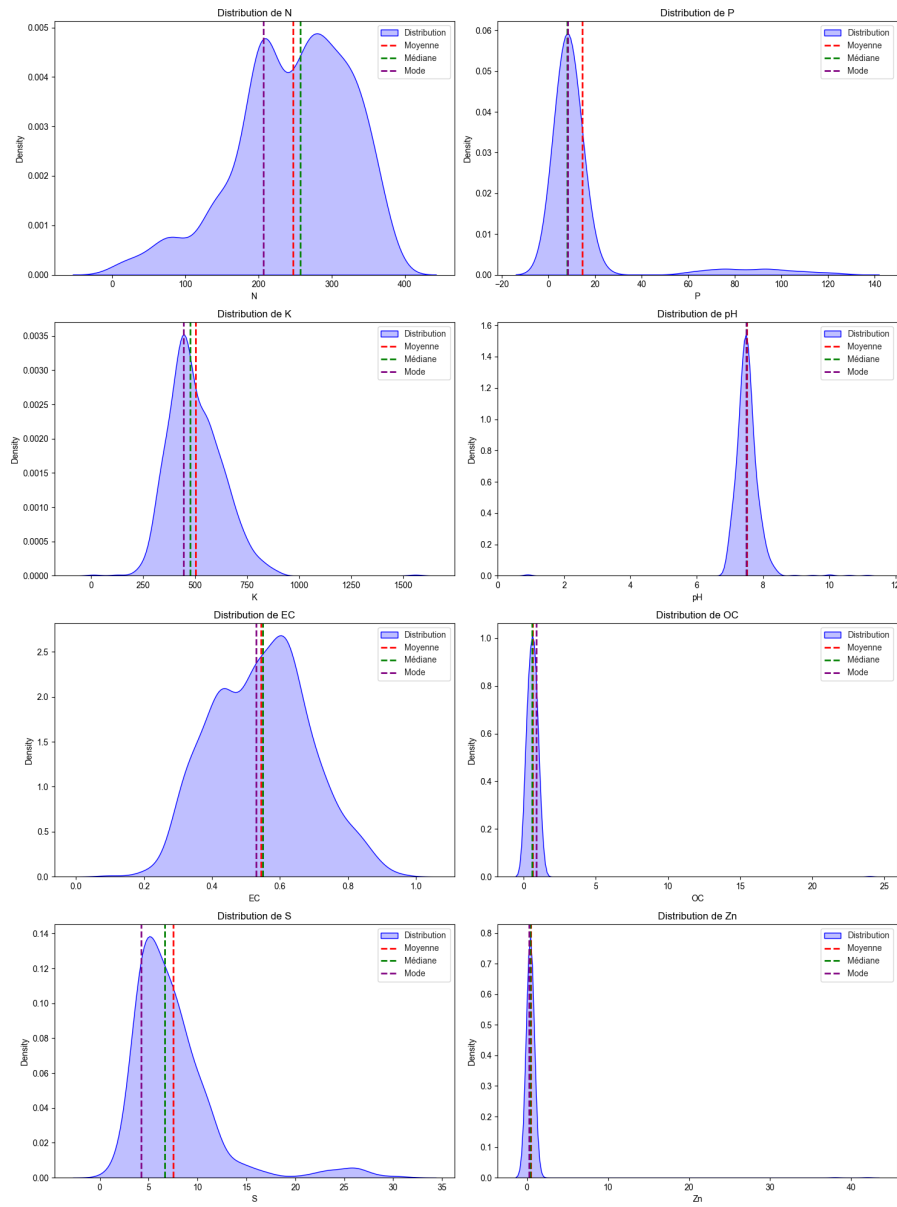


FIGURE 1 – Graphe Densité de N, K, E, C, S, P, Pc, Oc et Zn

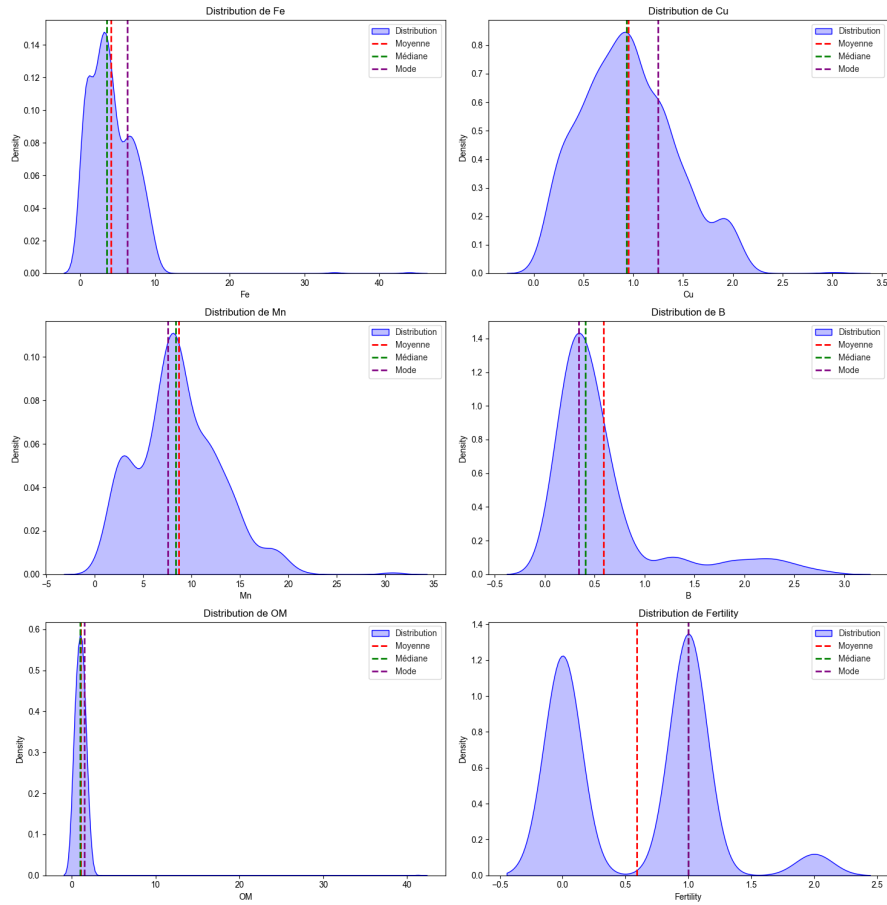


FIGURE 2 – Graphe Densité de MN, B, OM, Fertility, Fe et Cu

Analyse

Les graphes de densité examinés révèlent une proximité remarquable entre la moyenne, la médiane et le mode pour les attributs K, EC, Mn et OC, indiquant des distributions quasiment normales pour ces paramètres. En ce qui concerne le pH, Zn et OM, l'égalité entre la moyenne, la médiane et le mode souligne des distributions parfaitement symétriques. Pour les autres variables étudiées, les distributions ne montrent pas de symétrie prononcée.

Conclusion

Ces observations mettent en lumière une homogénéité des données pour K, EC, Mn et OC, qui reflète une certaine régularité ou constance de ces caractéristiques du sol. La distribution symétrique des valeurs de pH, Zn et OM autour de la moyenne témoigne d'une homogénéité des échantillons pour ces paramètres. En revanche, l'absence de symétrie pour les autres attributs suggère des variations plus marquées dans la distribution des données de ces paramètres du sol.

3.2 Construction des boîtes à moustache pour la détections des données aberrante

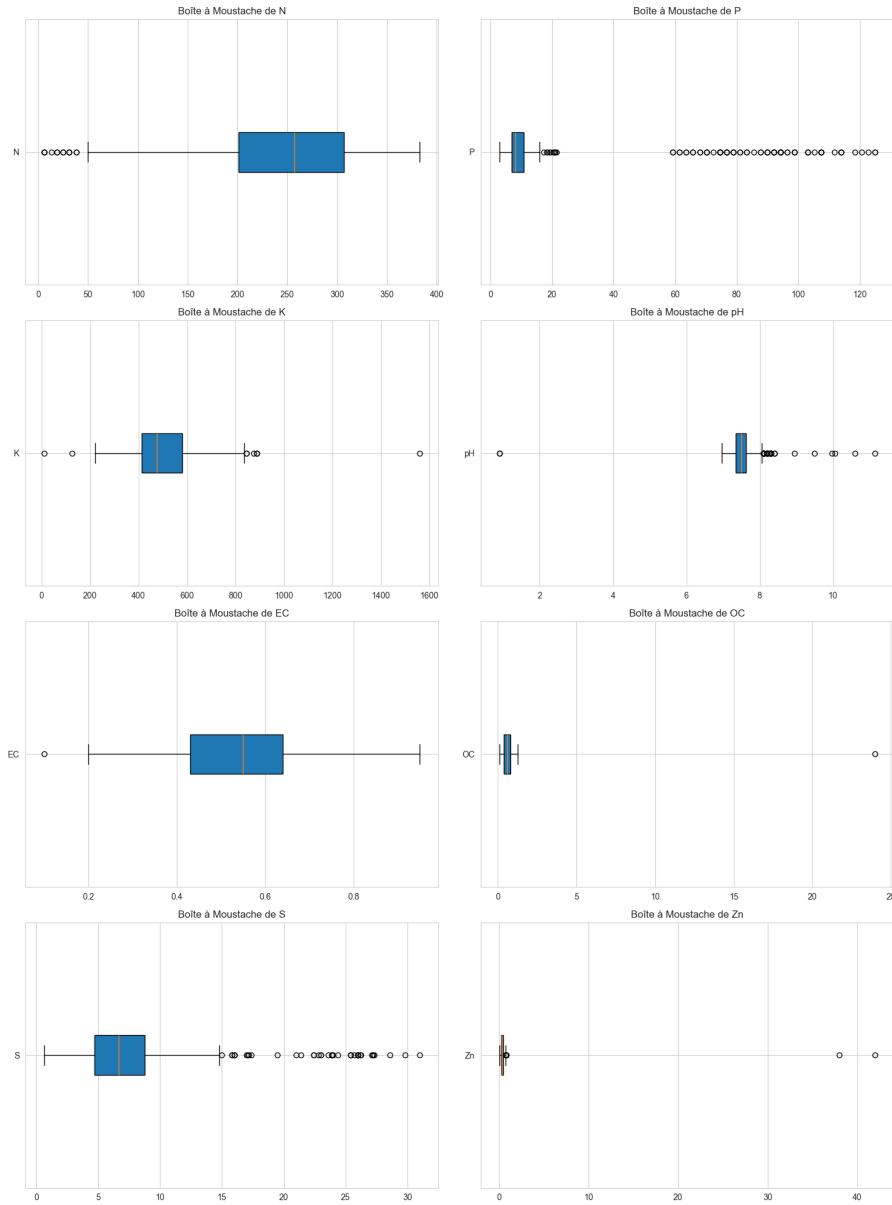


FIGURE 3 – Boîte à moustache

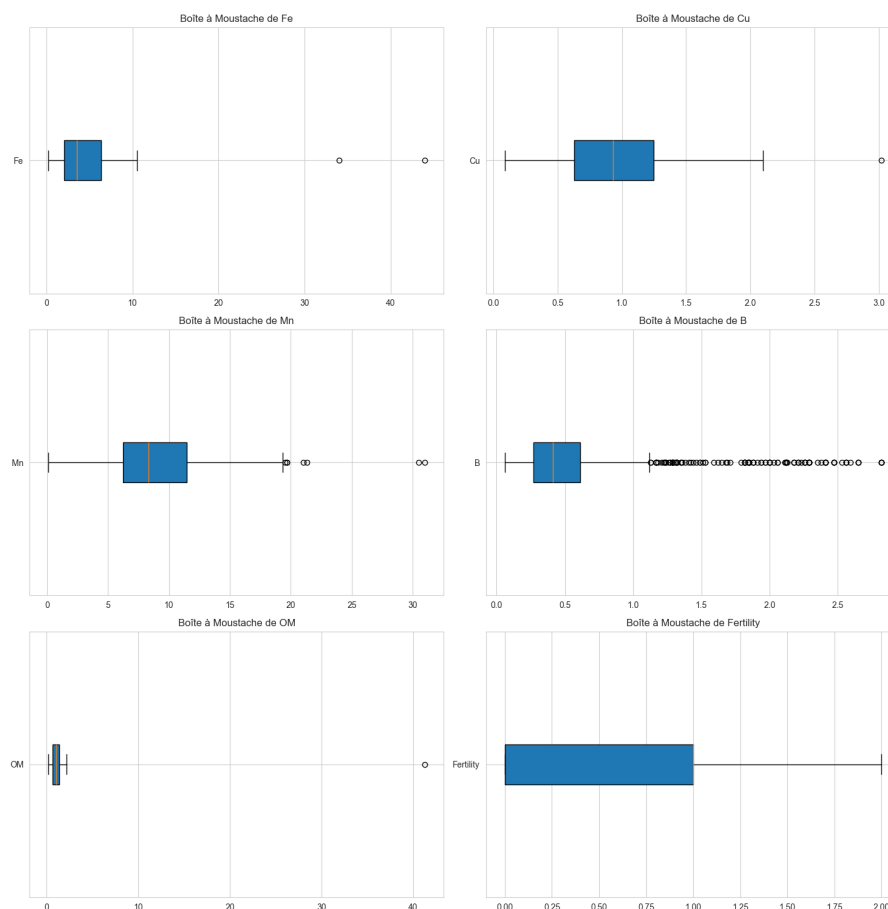


FIGURE 4 – Boîte à moustache (suite)

Analyse

L'examen des boîtes à moustaches révèle la présence de valeurs aberrantes pour chaque attribut étudié. On observe que certains attributs, tels que N, K, pH, S, Mn et B, affichent un nombre relativement plus important de ces valeurs atypiques. À l'opposé, les attributs tels que EC, Zn, Fe et MO se distinguent par leur faible nombre de valeurs aberrantes. La symétrie des distributions peut être évaluée à travers le positionnement de la médiane (trait orange) : une médiane centrée dans la boîte indique une distribution symétrique.

Conclusion

Il est impératif que la gestion et l'analyse des données prennent en considération ces disparités observées dans la fréquence des valeurs aberrantes, particulièrement pour les attributs qui démontrent une plus grande volatilité. Cette approche pourrait guider les études futures ainsi que les mesures correctives nécessaires pour garantir une compréhension exacte des propriétés du sol étudié. Il convient de réfléchir à des méthodes efficaces pour traiter ces valeurs atypiques, sujet qui sera exploré prochainement.

3.3 Construction des histogrammes des données

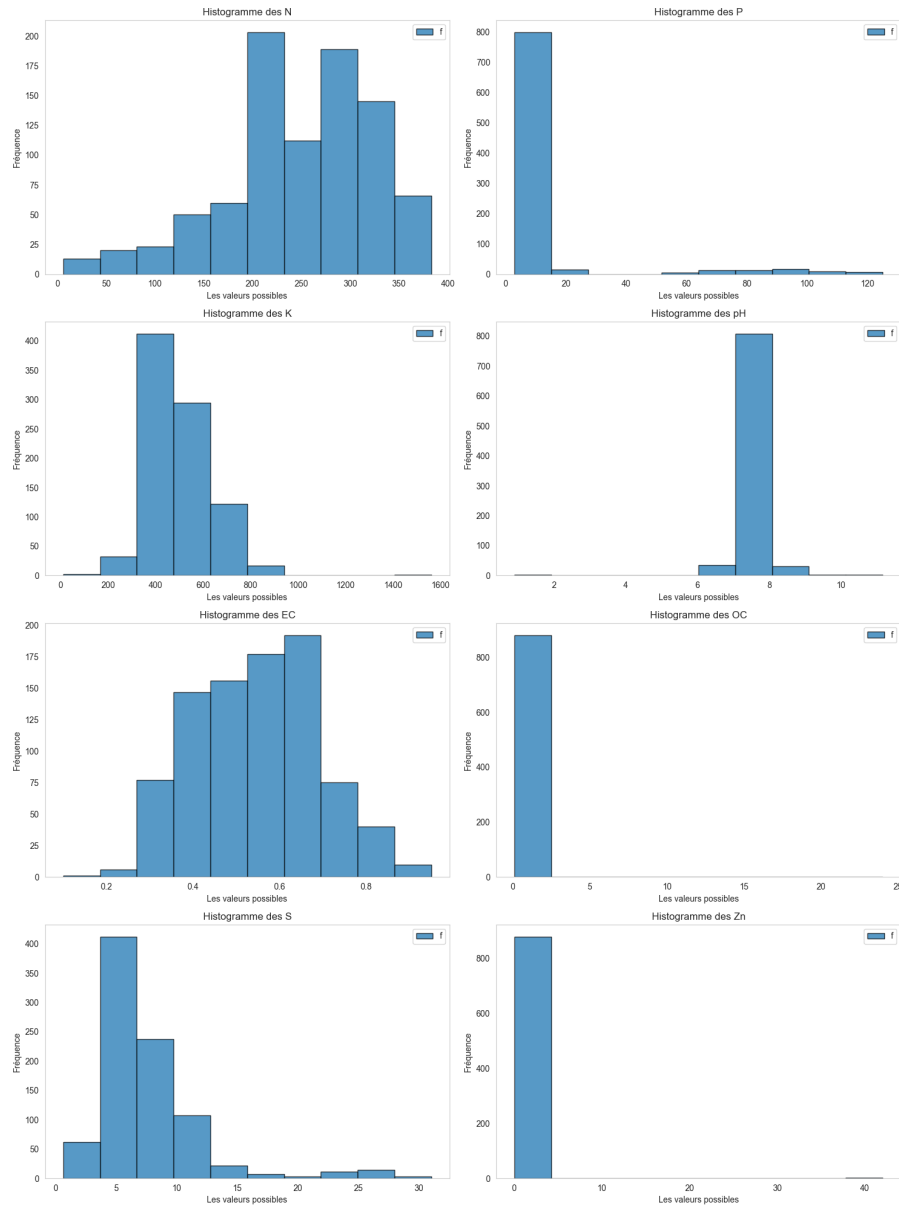


FIGURE 5 – Histogrammes

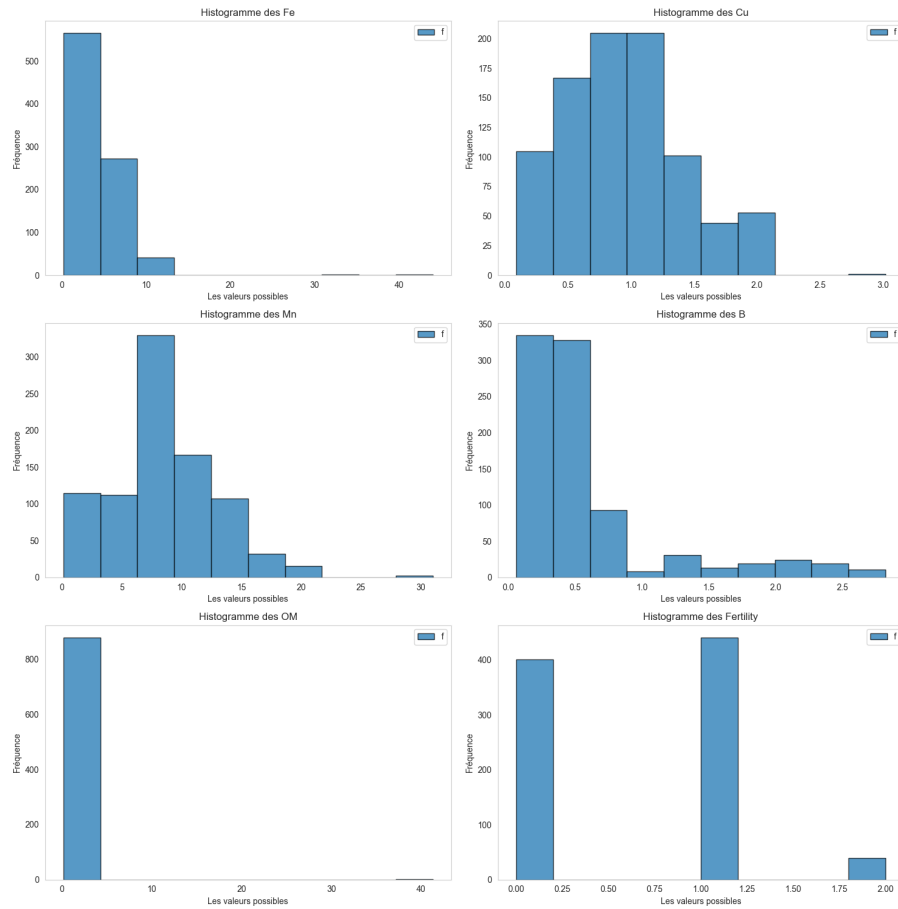


FIGURE 6 – Histogrammes (suite)

Les histogrammes de données offrent une analyse polyvalente, apportant divers éclairages :

- **Tendance et symétrie des données** : Les histogrammes permettent une évaluation rapide des tendances, du mode et de la symétrie des données. Par exemple, la confirmation de la symétrie pour pH, OM et Zn souligne une cohérence notable dans ces données.
- **Centrage et dispersion** : Ils fournissent une représentation visuelle claire du centrage (moyenne, médiane) ainsi que de la dispersion des données, offrant une vue immédiate de la variabilité des valeurs analysées.
- **Identification des anomalies** : Ces graphiques permettent de distinguer les valeurs rares, les plus fréquentes et les anomalies. Par exemple, ils mettent en lumière une présence très limitée de données à des valeurs spécifiques, comme 30 pour Fe et 3 pour Cu, soulignant des points singuliers dans le jeu de données.

3.4 Construction des diagrammes de dispersion des données

Pour la construction des diagrammes de dispersion, nous avons calculé la matrice de corrélation suivante :

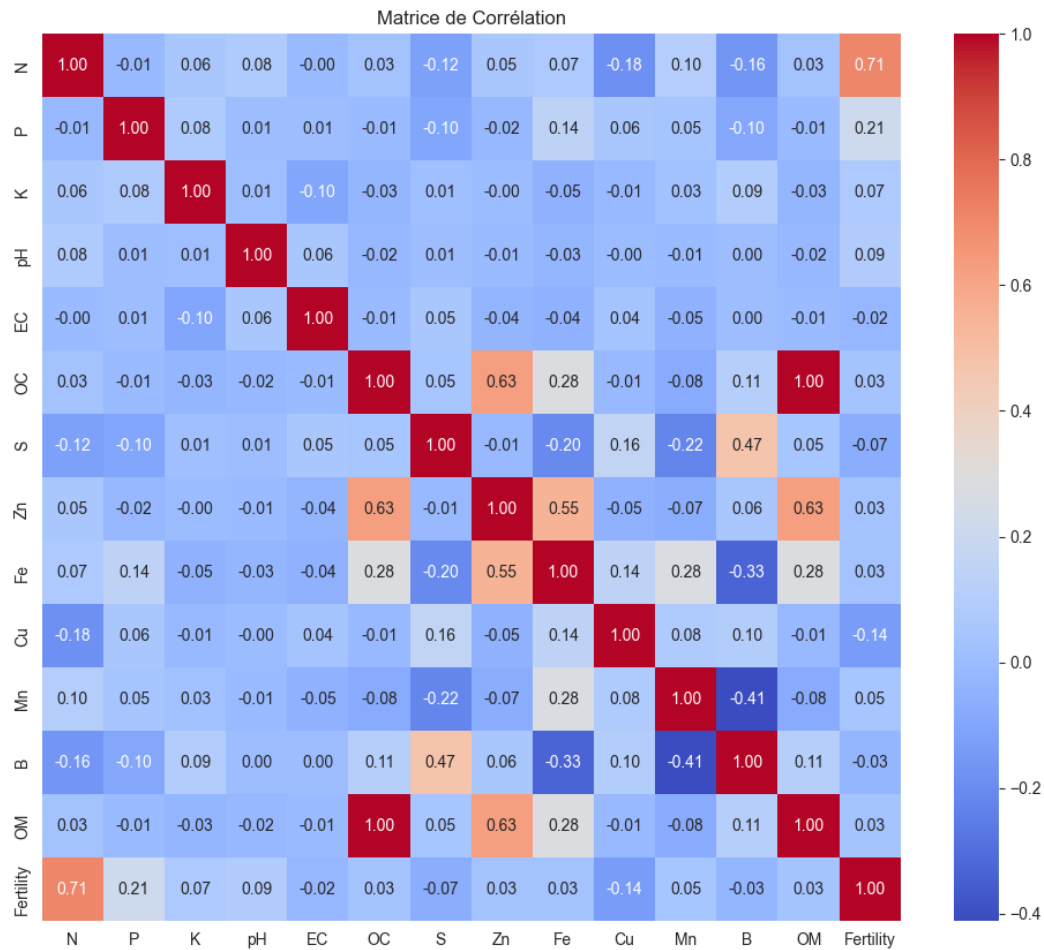


FIGURE 7 – La matrice de corrélation de dataset-1-

La matrice de corrélation est une table qui montre les coefficients de corrélation entre de multiples variables.

Le coefficient de corrélation "r" mesure la force et la direction de la relation linéaire entre deux variables. si $r=1$ indique une corrélation positive parfaite, ce qui signifie que les variables évoluent ensemble dans la même direction. sinon, si $r=-1$ indique une corrélation négative parfaite, ce qui signifie que les variables évoluent ensemble dans des directions opposées. sinon ($r=0$) indique l'absence de corrélation linéaire entre les variables.

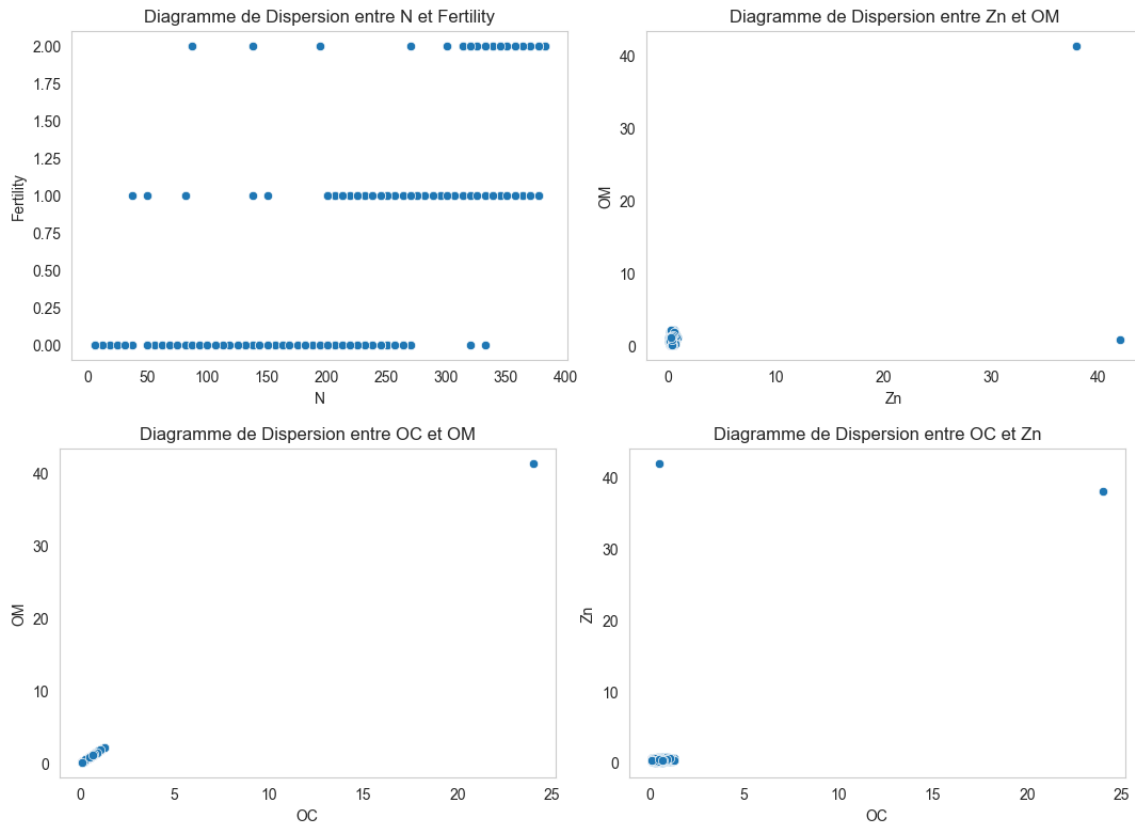


FIGURE 8 – La matrice de corrélation de dataset-1-

Les quatre graphiques de la [Figure 8] de dispersion montrent peu ou pas de corrélation linéaire forte entre les variables examinées (N et fertilité, Zn et MO, OC et MO, OC et Zn). Les points sont soit agglomérés près de l'origine, soit distribués en bandes horizontales, ce qui suggère des relations complexes ou des mesures discrètes. Des valeurs aberrantes sont présentes et pourraient influencer toute analyse de corrélation. En général, les données ne suggèrent pas de tendances linéaires simples entre les variables. Une analyse statistique plus approfondie serait nécessaire pour comprendre pleinement ces relations.

4 Prétraitement

4.1 Traitement des valeurs manquantes et aberrantes

4.1.1 Choix de la méthode de remplacement des valeurs manquantes

Les données manquantes peuvent avoir un effet important sur les analyses de données. Elles peuvent restreindre la taille de l'échantillon, entraîner une perte d'informations cruciales et affaiblir la robustesse des modèles utilisés dans le processus de data mining. Voici comment les valeurs manquantes sont présentées dans notre ensemble de données...

N	0	Fe	0
P	2	Cu	1
K	0	Mn	0
pH	0	B	0
EC	0	OM	0
OC	1	Fertility	0
S	0	dtype: int64	
Zn	0		

FIGURE 9 – Les valeurs nulles pour chaque attributs

Analyse

D'après l'observation de la figure , il est remarquable que le nombre de valeurs manquantes pour chaque attribut n'est pas considérablement élevé : 2 pour P, 1 pour OC et Cu. Pour remédier à ces valeurs manquantes, trois approches sont envisagées...

1. **Suppression** des Valeurs Manquantes
2. Remplacement par la **Moyenne**
3. Remplacement par la **mediane**

Conclusion

Le choix parmi ces trois méthodes dépend étroitement des objectifs spécifiques de l'analyse ainsi que de la nature des données. La suppression des lignes pourrait être considérée si le nombre de valeurs manquantes est limité et n'affecte pas de manière significative la taille totale de l'ensemble de données. Cependant, lorsque les valeurs manquantes sont réparties de manière uniforme et que leur suppression risque de introduire un biais, le remplacement par la moyenne ou la médiane pourrait être une alternative plus adaptée pour maintenir l'intégrité des données.

4.1.2 Choix de la méthode de traitement des valeurs aberrantes

Les valeurs aberrantes, souvent désignées sous le terme d'outliers, peuvent considérablement affecter les analyses de données. Elles influent sur les mesures de tendance centrale, impactent la dispersion des données et peuvent fausser les corrélations observées. Suite à leur élimination, tous les calculs précédents doivent être réitérés afin d'obtenir des résultats plus précis dans ce contexte.

Le repérage des outliers a déjà été effectué via les boxplots lors de la phase de construction des boîtes à moustaches pour détecter ces données aberrantes. Dans ce rapport, deux méthodes sont proposées pour les traiter : le calcul de la moyenne et l'utilisation de la médiane.

Voici comment ces deux mesures peuvent être utilisées pour traiter les outliers...

- Traitement des outliers par la moyenne** : On comparant avec les boites a moustache avant le traitement des outliers dans la [Figure 3 et 4], cette méthode élimine bien les outliers de N, P, K, EC, OC, Zn, Fe, Cu, OM. Il reste cependant quelques-uns dans le pH et le B, mais le nombre n'est pas bien réduit pour S et Mn.

Et voici les boites a moustaches après le traitement par la moyenne [Figure 10 et 11] :

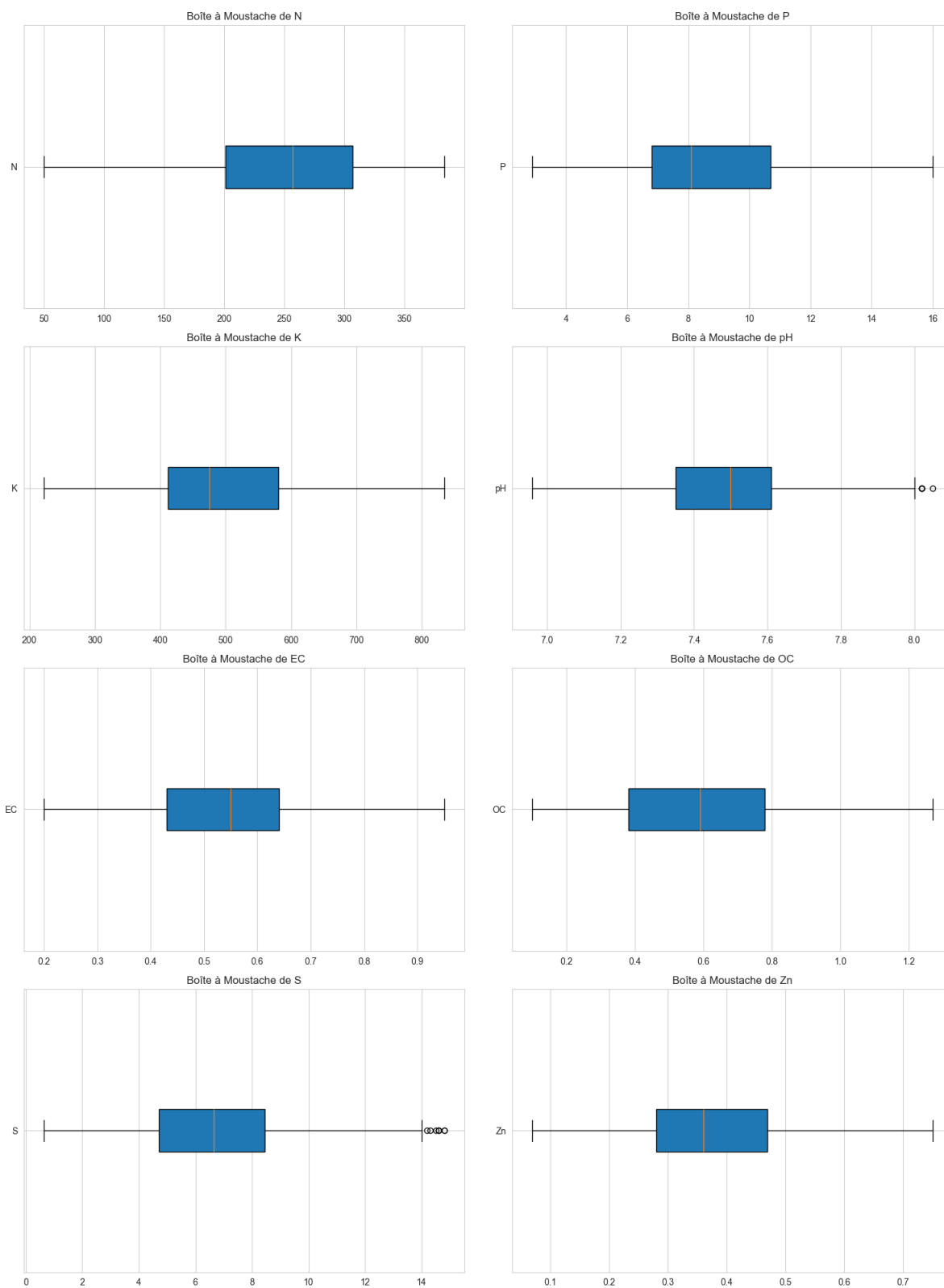


FIGURE 10 – Boîte a moustache sans valeurs aberrantes (moyenne)

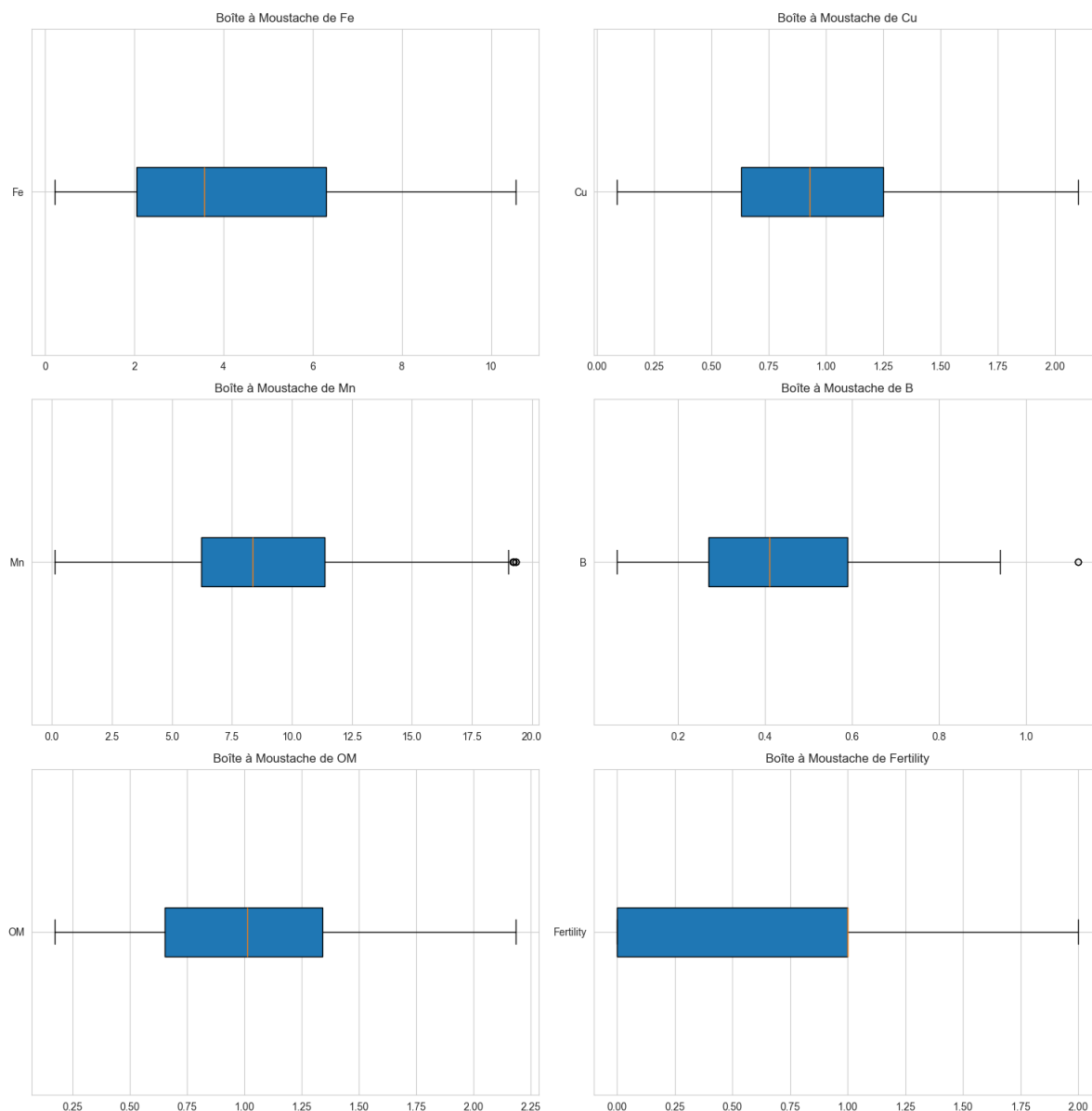


FIGURE 11 – Boîte a moustache sans valeurs aberrantes (suite - moyenne)

II. **Traitement des outliers par la médiane :** On comparant avec les boîtes a moustache avant le traitement des outliers dans la [Figure 3 et 4], cette méthode élimine bien les outliers de N, K, Ec, OC, Fe, Cu, OM. Il reste cependant quelques-uns dans le pH et le Zn, mais le nombre n'est pas réduit pour P, S, Mn et B.

Et voici les boîtes a moustaches après le traitement par la moyenne [Figure 12 et 13] :

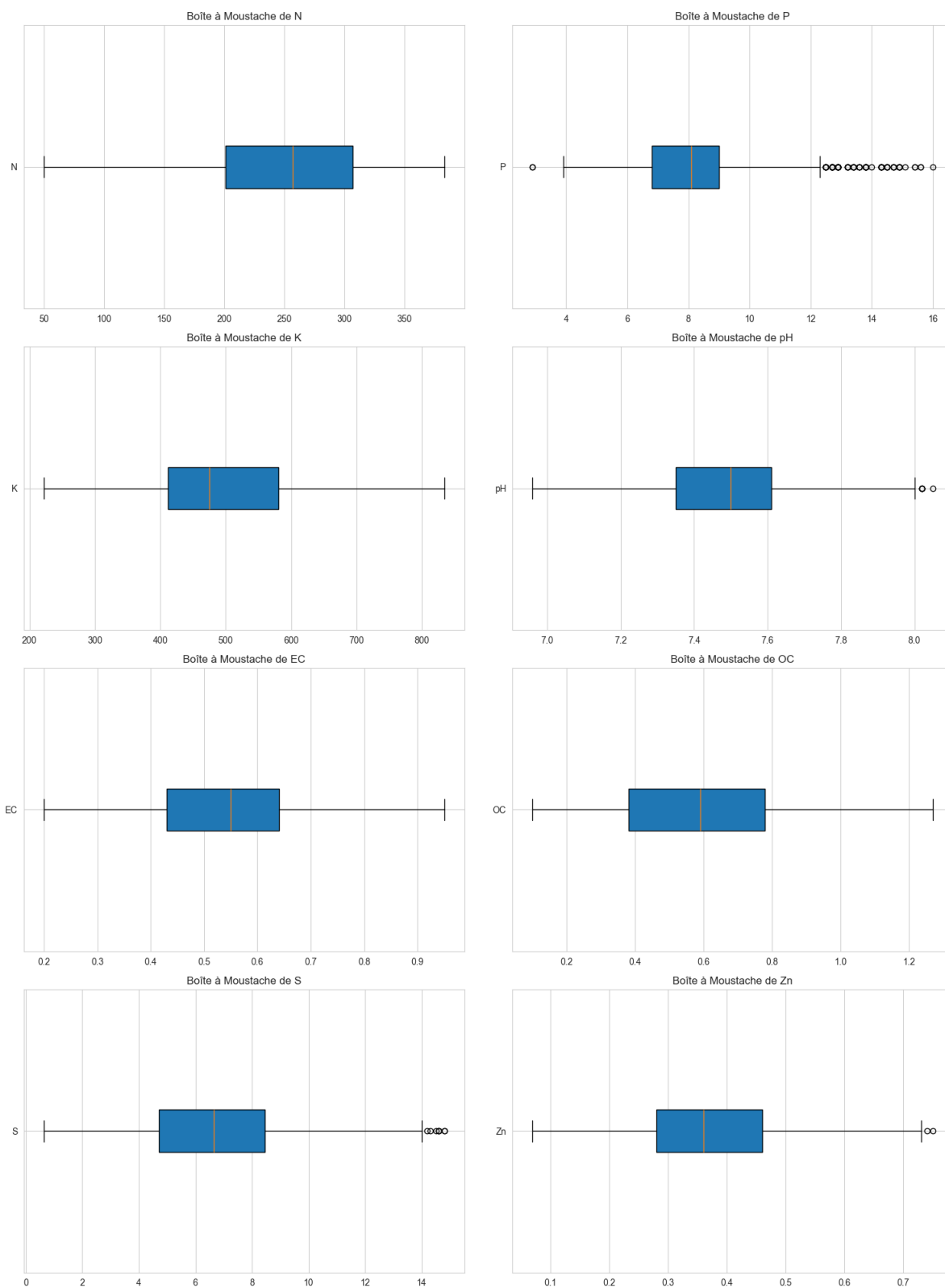


FIGURE 12 – Boîte a moustache sans valeurs aberrantes (mediane)

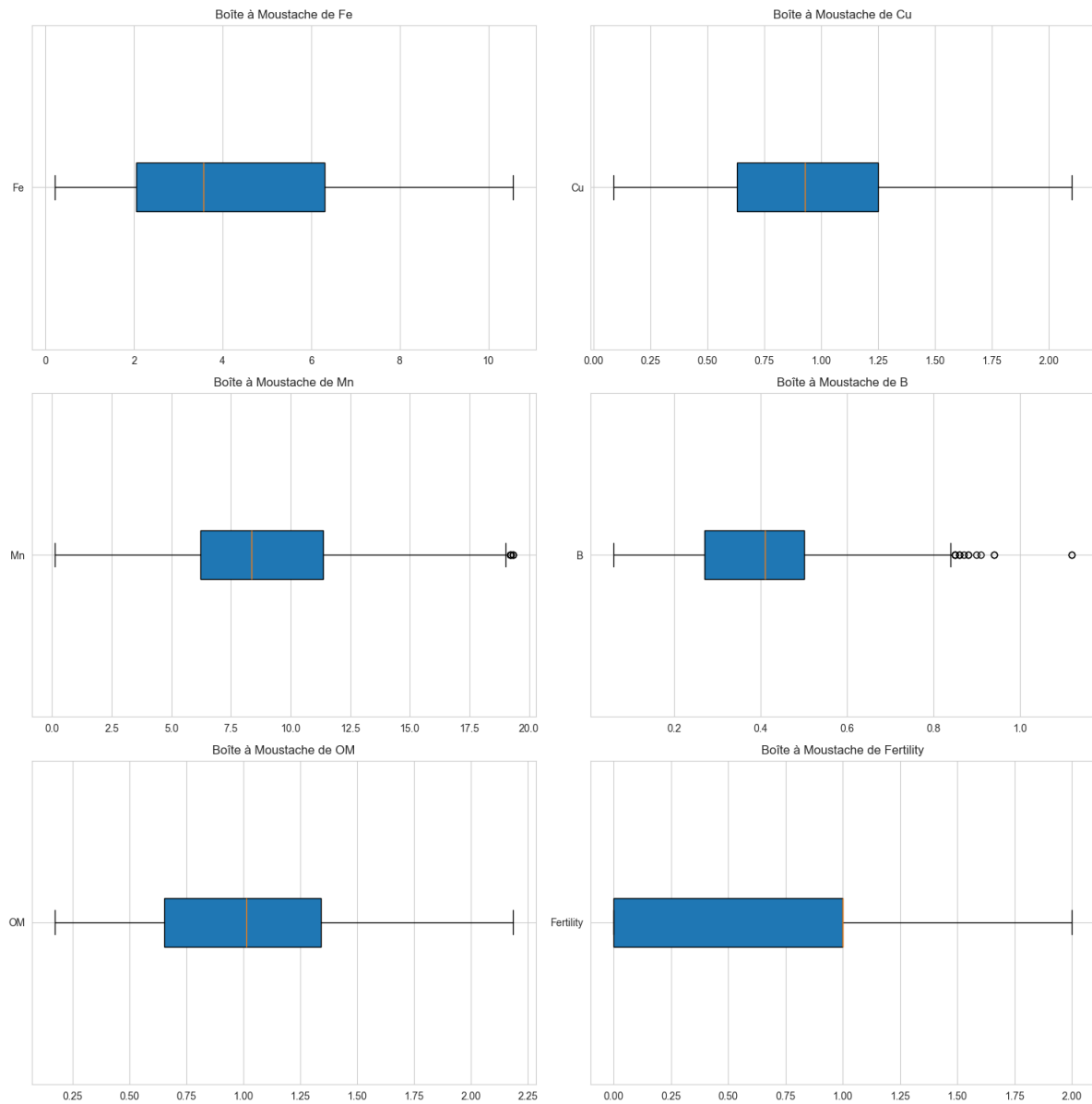


FIGURE 13 – Boîte a moustache sans valeurs aberrantes (suite - mediane)

Après l'application des deux méthodes, il est clair que chacune parvient à éliminer les outliers, mais pas de manière exhaustive. Quelques-uns persistent, et il est à noter que la méthode basée sur la moyenne a mieux traité les outliers que celle utilisant la médiane. En conséquence, nous recommandons d'opter pour la première méthode, celle de la moyenne, et de supprimer manuellement les outliers restants tant que leur nombre demeure limité. On prend l'exemple de la colonne B en éliminant les outliers manuellement [Figure 14] :

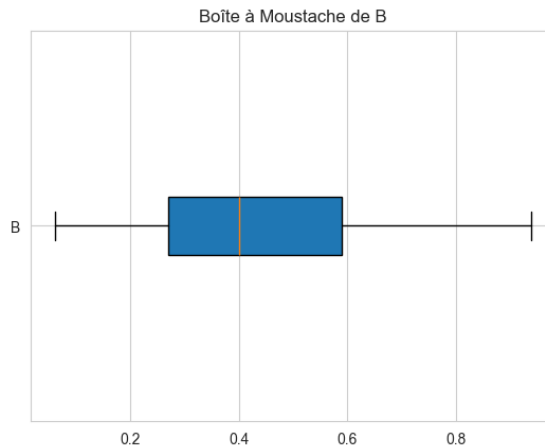


FIGURE 14 – Élimination des valeurs aberrantes de B

Conclusion

En définitive, le choix entre la moyenne et la médiane dépend étroitement de la nature particulière de vos données et des objectifs définis pour votre analyse. Il peut être bénéfique d'explorer ces deux approches et de comparer les résultats afin de prendre une décision éclairée, comme nous l'avons fait ici en optant pour **l'utilisation de la moyenne**.

4.2 Réduction des données (élimination des redondances)

En visant à simplifier les données tout en préservant les informations cruciales, l'objectif principal est de réduire la complexité. Dans ce contexte, éliminer les redondances implique la suppression des données en double ou superflues présentes dans l'ensemble de données.

Les dimensions du jeu de données :

- (879, 14)

4.2.1 verticales

- (879, 12)

Identifiez et supprimez les variables qui présentent une corrélation élevée entre elles, car elles peuvent apporter des informations similaires. Par exemple $r=1$ entre OC et OM donc on peut **supprimer les 2 colonnes OM et OC**.

4.2.2 horizontales

- (876, 12)

On remarque que le nombre de lignes a diminué de 3 lignes.

4.3 Normalisation des données

La normalisation des données consiste à ajuster l'échelle des variables au sein d'un ensemble de données. Son objectif principal est de rendre les données comparables et de simplifier l'analyse en assurant que chaque variable contribue de manière équitable aux calculs, indépendamment de leurs unités d'origine ou de leurs échelles respectives.

4.3.1 Méthode Min-Max

Avec la formule suivante, on peut appliquer cette méthode, et on peut prendre par exemple le max = 1 et le min = 0, pour avoir des valeurs entre 0 et 1 :

$$X_{\text{norm}} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

	N	P	K	pH	EC	S	Zn	Fe	Cu	Mn	B	Fertility
0	0.264264	0.435115	0.552288	0.458716	0.560000	0.371469	0.250000	0.009671	0.338308	0.447683	0.056818	0.0
1	0.489489	0.351145	0.189542	0.605505	0.733333	0.486496	0.338235	0.062863	0.721393	0.144716	0.602363	0.0
2	0.339339	0.511450	0.810458	0.577982	0.413333	0.964689	0.338235	0.062863	0.736318	0.134826	0.602363	0.0
3	0.321321	0.297710	0.413399	0.623853	0.506667	0.486496	0.397059	0.031915	0.716418	0.132223	0.602363	0.0
4	0.660661	0.534351	0.362745	0.614679	0.266667	0.788136	0.264706	0.053191	0.796020	0.120770	0.602363	1.0

FIGURE 15 – Portion Résultat Normalisation Min-Max

4.3.2 Méthode z-score

Cette méthode transforme les données pour qu'elles aient une moyenne de 0 et un écart-type de 1. La formule est la suivante :

$$Z = \frac{X - \mu}{\sigma}$$

	N	P	K	pH	EC	S	Zn	Fe	Cu	Mn	B	Fertility
0	-1.550684	-0.092849	0.523096	-0.103073	0.536178	-0.346445	-1.154531	-1.443713	-0.393436	0.030252	-1.613986	0.0
1	-0.512595	-0.468777	-1.354580	0.604263	1.459291	0.290286	-0.650455	-1.232806	1.280454	-1.404217	0.941309	0.0
2	-1.204654	0.248904	1.859460	0.471638	-0.244918	2.937315	-0.650455	-1.232806	1.345670	-1.451047	0.941309	0.0
3	-1.287701	-0.708003	-0.195834	0.692680	0.252143	0.290286	-0.314404	-1.355515	1.258715	-1.463370	0.941309	0.0
4	0.276352	0.351430	-0.458032	0.648472	-1.026013	1.960006	-1.070518	-1.271153	1.606536	-1.517594	0.941309	1.0

FIGURE 16 – Portion Résultat normalisées Z-score

Deuxième partie

Analyse supervisée

1 Objectif

L'objectif principal de ce projet est d'appliquer des techniques d'analyse supervisée pour la classification de l'attribut "Fertility" dans un dataset de sol. En utilisant des méthodes de machine learning, nous cherchons à prédire si un échantillon de sol est faiblement, moyennement ou fortement fertile. Cette partie détaille le processus d'implémentation de trois algorithmes de classification : KNN, Decision Trees, et Random Forest, et analyse leurs performances respectives.

2 Méthodologie :

2.1 Préparation des Données :

- La préparation des données est une étape fondamentale de notre analyse supervisée. Notre dataset, qui comprend des échantillons de sol avec une variable cible "Fertility" divisée en trois classes, a été divisé en données d'apprentissage et de test. Nous avons attribué **80%** du dataset à l'apprentissage et 20% au test. La division a été effectuée de manière stratifiée pour maintenir une représentation proportionnelle des classes de fertilité, assurant ainsi une distribution équilibrée et évitant les biais.
- Avant la séparation, les données ont été soigneusement nettoyées pour traiter les valeurs aberrantes. Pour ce faire, nous avons utilisé le median de chaque colonne. On a éliminé les valeurs nulles, supprimé les redondances tant au niveau des colonnes (OC et OM) et des lignes.
- La stratification a été réalisée en utilisant la fonction **train_test_split** de la bibliothèque **sklearn**, avec le paramètre **stratify** défini sur la variable cible y. Cette technique assure que la proportion des classes de fertilité est constante entre les ensembles d'apprentissage et de test, ce qui est crucial pour éviter tout biais dû à une distribution inégale des classes. **Par exemple**, si une classe était sous-représentée dans l'ensemble de test, les modèles pourraient ne pas être évalués de manière appropriée pour leur capacité à identifier cette classe.
- Une vérification finale a confirmé la répartition adéquate des caractéristiques et des étiquettes ainsi que les proportions correctes entre les ensembles d'apprentissage et de test. Cette étape détermine la qualité de l'entraînement des modèles et l'exactitude des prédictions subséquentes.

2.2 Sélection des Modèles de Classification :

Trois algorithmes réputés ont été retenus pour leur performance robuste et leur flexibilité face à divers problèmes de classification.

1. **K plus proches voisins (KNN)** : KNN est un algorithme d'apprentissage supervisé qui classe un nouvel échantillon en se basant sur la majorité des classes de ses voisins les plus proches dans l'espace des caractéristiques. [2]

2. **Forêt aléatoire (Random Forest)** : La forêt aléatoire est un ensemble d'arbres de décision où chaque arbre est construit à partir d'un échantillon de données et fournit une prédiction. La prédiction finale est obtenue par vote majoritaire ou moyenne des prédictions individuelles des arbres.
3. **Arbres de décision (Decision Trees)** : Les arbres de décision organisent les données sous forme d'une structure arborescente. Chaque nœud représente une caractéristique, chaque branche une décision basée sur cette caractéristique, et chaque feuille du arbre une prédiction ou une valeur. Ils sont construits par division itérative des données selon les caractéristiques les plus importantes.

2.3 Application des Modèles :

2.3.1 Méthodes d'application des trois algorithmes sur les instances du dataset :

L'application des modèles sur les instances du dataset est une étape clé du processus de machine learning, qui permet d'évaluer la capacité des algorithmes à généraliser à partir des données d'entraînement vers des données non vues. Dans notre cas, nous avons choisi trois algorithmes de classification : K-Nearest Neighbors (KNN), Decision Trees et Random Forest. Voici une explication détaillée de la manière dont chaque algorithme a été appliqué :

1. L'implémentation de KNN :

Algorithm 1 KNN algorithm

Input: training_set, test_x, k, distance_function

Output: predictions

predictions \leftarrow Empty list

for *each instance in test_x* **do**

 prediction \leftarrow predict_knn(training_set, instance, k, distance_function) Add prediction to predictions list

end

return predictions

L'élaboration de l'algorithme KNN (K plus proches voisins) pour la classification implique l'utilisation de plusieurs fonctions clés. La collecte des données et leur prétraitement sont réalisés à l'aide de fonctions dédiées. La sélection de la mesure de distance, effectuée par des fonctions telles que "**trier_selon_distance**", permet de calculer les distances entre les instances du jeu de données. La fonction "**predict_knn**" utilise ces distances pour prédire la classe d'une nouvelle instance en identifiant les K voisins les plus proches. En outre, des fonctions d'évaluation, d'optimisation des hyperparamètres et de déploiement peuvent être utilisées pour évaluer, ajuster et intégrer le modèle KNN dans des systèmes, offrant ainsi une approche flexible et adaptable pour la classification.

2. L'implémentation de Decision Trees :

Algorithm 2 DecisionTrees algorithm

Input: training_set, test_set**Output:** predictionstree \leftarrow build_tree(training_set) predictions \leftarrow Empty list**for** each instance in test_set **do**| prediction \leftarrow Get classification result from tree for instance Add prediction to predictions list**end****return** predictions

L'implémentation de l'algorithme de l'arbre de décision repose sur plusieurs fonctions clés. **class_counts(rows)** compte les occurrences de chaque classe dans un jeu de données, **partition(rows, column, value)** divise les données en fonction de la valeur d'une colonne spécifique, tandis que **gini(rows)** calcule l'indice de Gini pour évaluer l'impureté des données. **find_best_split(rows)** identifie la meilleure division basée sur le gain d'information et l'indice de Gini, **build_tree(rows)** construit récursivement l'arbre en sélectionnant les meilleures divisions, et **classify(row, node)** assigne une classe à une instance donnée en naviguant dans l'arbre. Enfin, **DecisionTrees(training_set, test_set)** utilise l'ensemble d'entraînement pour construire l'arbre de décision et prédire les classes des données de test en parcourant l'arbre pour chaque instance de test. Ces fonctions interagissent pour créer un arbre de décision en utilisant le critère du gain d'information pour les divisions et pour classer de nouvelles instances en fonction de ces divisions.

3. L'implémentation de Random Forest :

Algorithm 3 RandomForest

Input: training_set, test_set, n_trees**Output:** predictionsforest = random_forest_train(training_set, n_trees) predictions = tableau_vide **for** chaque instance dans test_set **do**

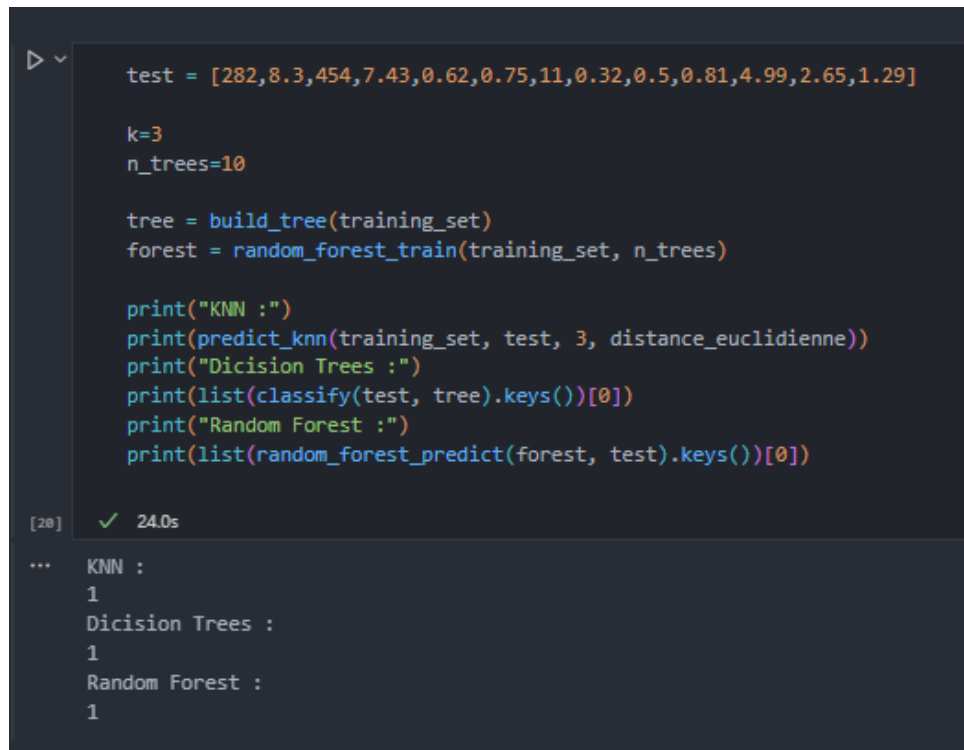
| prediction = random_forest_predict(forest, instance) Ajouter prediction à predictions

end**return** predictions

L'algorithme de la forêt aléatoire repose sur plusieurs fonctions essentielles. **bootstrap_sample(data)** génère un échantillon bootstrap à partir des données en sélectionnant aléatoirement des échantillons avec remplacement. **random_forest_train(data, n_trees)** entraîne un ensemble de n_trees arbres de décision en utilisant l'échantillonnage bootstrap, construisant ainsi plusieurs arbres basés sur des sous-ensembles aléatoires des données d'entraînement. **random_forest_predict(trees, row)** prédit la classe d'une nouvelle instance en utilisant l'ensemble d'arbres entraînés, combinant les prédictions pour obtenir une prédiction finale par vote majoritaire. La fonction principale, **RandomForest(training_set, test_set, n_trees)**, crée une forêt aléatoire en utilisant **random_forest_train** pour l'entraînement et **random_forest_predict** pour prédire les classes des données de test. Cette approche par ensemble d'arbres de décision, où chaque

arbre contribue à la prédiction finale, permet généralement une amélioration des performances grâce à la diversité des modèles et à l'agrégation des prédictions.

Application des trois algorithmes sur les instances du dataset :



```
test = [282,8.3,454,7.43,0.62,0.75,11,0.32,0.5,0.81,4.99,2.65,1.29]

k=3
n_trees=10

tree = build_tree(training_set)
forest = random_forest_train(training_set, n_trees)

print("KNN :")
print(predict_knn(training_set, test, 3, distance_euclidienne))
print("Dicision Trees :")
print(list(classify(test, tree).keys())[0])
print("Random Forest :")
print(list(random_forest_predict(forest, test).keys())[0])

[20] ✓ 24.0s

... KNN :
1
Dicision Trees :
1
Random Forest :
1
```

FIGURE 17 – Enter Caption

2.3.2 Illustration par des Exemples de l'Application des Modèles

Dans notre projet, l'application des trois modèles de classification a été concrètement illustrée par l'exemple suivant :

Exemple de Test

Nous avons pris un échantillon de test avec les caractéristiques suivantes : **[282, 8.3, 454, 7.43, 0.62, 0.75, 11, 0.32, 0.5, 0.81, 4.99, 2.65, 1.29]**. Cet échantillon représente une combinaison de divers paramètres mesurés dans le sol, comme le pH, la teneur en minéraux, etc.

Application des Modèles

1. K-Nearest Neighbors (KNN) :

- Le modèle KNN a été appliqué avec $k = 3$, cherchant les trois voisins les plus proches dans l'espace des caractéristiques.

- La fonction **predict_knn** a été utilisée pour prédire la classe de fertilité de l'échantillon test, en utilisant la distance euclidienne comme mesure.

2. Decision Trees :

- Un arbre de décision a été construit à partir de l'ensemble d'entraînement.
- La fonction **classify** a été utilisée pour déterminer la classe de fertilité de l'échantillon test, en parcourant l'arbre et en suivant les critères établis à chaque nœud.

3. Random Forest :

- Une forêt aléatoire comprenant $n_trees = 10$ arbres a été entraînée.
- La fonction **random_forest_predict** a été utilisée pour prédire la classe de l'échantillon test en agrégeant les prédictions de tous les arbres.

Résultats

L'application des modèles KNN, Decision Trees et Random Forest sur un échantillon de test a abouti à une prédiction unanime de la classe 1, démontrant une forte concordance entre les méthodes de classification malgré leurs approches variées, et suggérant une prédiction fiable pour la fertilité du sol de l'échantillon analysé.

3 Évaluation des Modèles

L'évaluation des modèles de classification dans notre projet s'est appuyée sur l'utilisation de matrices de confusion, qui fournissent un aperçu détaillé de la performance de chaque modèle. La fonction **matrice_confusion** a été définie pour automatiser ce processus.

3.1 Explication sur la manière de donner la matrice de confusion

Fonction de Matrice de Confusion

```
def matrice_confusion(y_test, predictions):
    classes = y_test.drop_duplicates()

    matrice = [[0 for _ in classes] for _ in classes]
    class_to_index = {cls: i for i, cls in enumerate(classes)}

    for r, p in zip(y_test, predictions):
        actual_index = class_to_index[r]
        predicted_index = class_to_index[p]
        matrice[actual_index][predicted_index] += 1

    return matrice
```

FIGURE 18 – Fonction de Matrice de Confusion

La fonction `matrice_confusion` prend en entrée les étiquettes réelles `y_test` et les prédictions générées par le modèle. Elle initie d'abord une matrice carrée basée sur le nombre de classes uniques dans `y_test`, puis parcourt toutes les prédictions et incrémente les cellules correspondantes dans la matrice. Les indices réels et prédits sont déterminés en mappant chaque classe à un indice numérique. Pour chaque paire de vraie étiquette et de prédiction, l'élément correspondant dans la matrice est augmenté, reflétant le nombre de prédictions correctes et incorrectes pour chaque classe.

Application de la Fonction

```
MCKNN = matrice_confusion(y_test, KNNpredictions)
MCDT = matrice_confusion(y_test, DTpredictions)
MCRF = matrice_confusion(y_test, RFpredictions)
MCKNN, MCDT, MCRF

[26]
... ([[72, 9, 0], [7, 80, 1], [0, 7, 1]],
      [[69, 12, 0], [8, 73, 7], [1, 2, 5]],
      [[69, 12, 0], [7, 71, 10], [1, 2, 5]])
```

FIGURE 19 – Application de la Fonction

Nous avons appliqué cette fonction pour générer des matrices de confusion pour chacun de nos modèles : KNN, Decision Trees et Random Forest. Les matrices de confusion résultantes sont :

- Matrice de Confusion pour KNN (**MCKNN**)
- Matrice de Confusion pour Decision Trees (**MCDT**)
- Matrice de Confusion pour Random Forest (**MCRF**)

Chaque matrice est un tableau à deux dimensions où l'élément $M[i][j]$ représente le nombre d'instances de la classe réelle i qui ont été prédites comme la classe j par le modèle. La diagonale principale de chaque matrice montre le nombre de prédictions correctes, tandis que les autres valeurs indiquent les différents types d'erreurs de classification.

Interprétation des Matrices

Par exemple, si nous considérons la matrice de confusion pour le modèle KNN, les éléments sur la diagonale principale indiquent les instances où le modèle a correctement prédit la classe de fertilité. Les éléments hors de la diagonale montrent où le modèle a fait des erreurs, soit en prédisant une fertilité plus élevée (faux positifs) ou plus faible (faux négatifs) que la classe réelle.

L'utilisation de ces matrices nous permet d'analyser en profondeur la performance de chaque modèle, en identifiant non seulement leur précision globale mais

aussi la nature des erreurs commises, ce qui est crucial pour affiner les modèles et améliorer les prédictions futures.

3.2 Méthodologie d'Évaluation des Modèles de Classification

Notre méthodologie d'évaluation des modèles de classification a été méticuleusement conçue pour mesurer la performance de manière détaillée et comparer efficacement les algorithmes utilisés. Nous avons calculé des mesures clés telles que l'exactitude, la spécificité, la précision, le rappel et le F-score pour chaque classe et globalement, en plus de mesurer le temps moyen d'exécution pour chaque modèle.

Fonctions de Performance Implémentées

Les fonctions suivantes ont été implémentées pour faciliter l'évaluation :

```
def accuracy(confusion_matrix):
    correct_predictions = sum(confusion_matrix[i][i] for i in range(len(confusion_matrix)))
    total_predictions = sum(sum(row) for row in confusion_matrix)
    accuracy = correct_predictions / total_predictions
    return accuracy

def specificite(confusion_matrix, classe):
    total = sum(sum(row) for row in confusion_matrix)
    TN_FP = total - sum(confusion_matrix[classe])
    TN = TN_FP - sum(confusion_matrix[i][classe] for i in range(len(confusion_matrix)))
    FP = sum(confusion_matrix[i][classe] for i in range(len(confusion_matrix))) - confusion_matrix[classe][classe]
    return TN / (TN + FP) if (TN + FP) > 0 else 0

def precision(confusion_matrix, classe):
    TP = confusion_matrix[classe][classe]
    FP = sum(confusion_matrix[i][classe] for i in range(len(confusion_matrix))) - TP
    return TP / (TP + FP) if (TP + FP) > 0 else 0

def rappel(confusion_matrix, classe):
    TP = confusion_matrix[classe][classe]
    FN = sum(confusion_matrix[classe]) - TP
    return TP / (TP + FN) if (TP + FN) > 0 else 0

def f_score(confusion_matrix, classe):
    prec = precision(confusion_matrix, classe)
    rapp = rappel(confusion_matrix, classe)
    return 2 * (prec * rapp) / (prec + rapp) if (prec + rapp) > 0 else 0
```

FIGURE 20 – Les fonctions d'évaluation

- **Exactitude** : Calculée en divisant le nombre de prédictions correctes par le nombre total de prédictions, cette mesure reflète la capacité globale du modèle à classer correctement les instances.
- **Spécificité** : Cette métrique indique la capacité du modèle à identifier correctement les véritables négatifs, ce qui est crucial dans des situations où les fausses alarmes sont coûteuses.
- **Précision** : Elle mesure la proportion des identifications positives réelles parmi toutes les identifications positives, soulignant l'exactitude du modèle lorsqu'il prédit une classe positive.
- **Rappel** : Aussi connu sous le nom de sensibilité, le rappel évalue la capacité du modèle à détecter toutes les instances positives réelles.
- **F-Score** : Le F-score est la moyenne harmonique de la précision et du rappel, offrant un équilibre entre ces deux métriques et est particulièrement utile lorsque les classes sont déséquilibrées.

Chaque fonction a été appliquée aux matrices de confusion générées par les prédictions des modèles, permettant une évaluation précise et détaillée de leur performance.

Application de l'Évaluation

```
def evaluation(modele, training_set, test_set, y_test, **kwargs):
    print(modele.__name__)
    start_time = time.time()
    predictions = modele(training_set, test_set, **kwargs)
    end_time = time.time()
    temps_execution = end_time - start_time

    cm = matrice_confusion(y_test, predictions)

    exactitude_globale = accuracy(cm)
    specificite_globale, precision_globale, rappel_globale, f_score_globale = 0, 0, 0, 0
    for classe in range(len(cm)):
        spec = specificite(cm, classe)
        prec = precision(cm, classe)
        rapp = rappel(cm, classe)
        fscr = f_score(cm, classe)

        specificite_globale += spec
        precision_globale += prec
        rappel_globale += rapp
        f_score_globale += fscr
    print(f"Classe {classe}: Specificite: {format(spec, '.2f')}, Precision: {format(prec, '.2f')},
    print(f"Exactitude Globale: {format(exactitude_globale, '.2f')}")
    print(f"Specificite Globale: {format(specificite_globale/len(cm), '.2f')}")
    print(f"Precision Globale: {format(precision_globale/len(cm), '.2f')}")
    print(f"Rappel Global: {format(rappel_globale/len(cm), '.2f')}")
    print(f"F-Score Global: {format(f_score_globale/len(cm), '.2f')}")
    print(f"Temps d'Execution: {format(temps_execution, '.2f')} secondes")
    print()

evaluation(KNN, training_set, test_set, y_test, k=3, distance_function=distance_euclidienne)
evaluation(DecisionTrees, training_set, test_set, y_test)
evaluation(RandomForest, training_set, test_set, y_test, n_trees=10)
```

FIGURE 21 – Application de l'Évaluation

La fonction d'évaluation exécute le modèle de classification, calcule les prédictions, mesure le temps d'exécution et applique les fonctions de performance pour générer les métriques d'évaluation. Les résultats ont été enregistrés pour chaque classe, ainsi qu'à l'échelle globale, fournissant une analyse complète de chaque modèle.

Résultats

Modèle	Classe	Spécificité	Précision	Rappel	F-Score	Temps d'Exécution
KNN	0	0.71	0.91	0.89	0.90	1.32 s
	1	0.78	0.83	0.91	0.87	
	2	0.99	0.50	0.12	0.20	
Decision Trees	0	0.67	0.88	0.85	0.87	2.89 s
	1	0.12	0.84	0.83	0.83	
	2	0.96	0.42	0.62	0.50	
Random Forest	0	0.70	0.91	0.89	0.91	23.76 s
	1	0.60	0.88	0.89	0.88	
	2	0.97	0.38	0.38	0.38	

TABLE 6 – Comparaison des performances des modèles de classification

3.3 Analyse et Comparaison des Performances des Modèles de Classification

En examinant le dernier tableau [Table 6] de résultats, nous observons les performances des modèles KNN, Decision Trees, et Random Forest sur deux classes distinctes. Voici une analyse comparative détaillée de chaque modèle :

KNN

- Classe 0 présente de fortes performances avec un F-Score de 0.90, suggérant une excellente capacité de classification.
- Classe 1 montre une spécificité négative, qui est probablement une erreur et doit être révérifiée.
- Classe 2 a un F-Score faible de 0.20, indiquant un besoin d'amélioration dans la classification positive.
- Le modèle est le plus rapide avec un temps d'exécution de 1.32 secondes.

Decision Trees

- Classe 0 affiche un F-Score robuste de 0.87, révélant une bonne performance globale.
- Classe 1 et 2 révèlent des faiblesses avec des scores de spécificité et de précision réduits.
- Avec un temps d'exécution de 2.89 secondes, ce modèle offre un compromis entre vitesse et précision.

Random Forest

- Les classes 0 et 1 bénéficient de scores F élevés, indiquant une classification fiable et équilibrée.
- Classe 2, bien que présentant une spécificité élevée, a besoin d'amélioration en précision et rappel.
- Avec 23.76 secondes, c'est le modèle le plus lent, ce qui peut être contraignant pour des applications nécessitant une réponse rapide.

3.4 Évaluation et Comparaison Globale

Le Random Forest montre la meilleure performance globale mais à un coût en temps de calcul plus élevé. Le KNN est préférable pour une réponse rapide malgré ses limitations sur certaines classes. Les arbres de décision sont recommandés pour un équilibre entre vitesse et précision, surtout si un réglage fin est possible.

La sélection du modèle optimal dépend des exigences spécifiques de l'application en termes de vitesse et de précision. Pour les tâches où la vitesse est primordiale, KNN serait le choix préférable. Si l'exactitude est la priorité, Random Forest serait plus approprié, tandis que Decision Trees offre un compromis efficace.

Troisième partie

Analyse non supervisée

1 Objectif

L'analyse de clustering est une technique essentielle d'apprentissage automatique non supervisé utilisée pour découvrir des structures cachées dans des ensembles de données non étiquetées. Dans notre étude, nous avons employé deux méthodes de clustering réputées : K-means, qui partitionne les données en K clusters basés sur la moyenne, et DBSCAN, qui exploite la densité pour identifier des regroupements et gérer le bruit. Alors que K-means excelle dans la gestion de grands volumes de données et nécessite une définition préalable du nombre de clusters, DBSCAN détecte des formes de clusters non standard sans avoir besoin de spécifier ce nombre. **L'objectif de cette partie est de mettre en lumière l'importance et l'applicabilité de ces algorithmes de clustering, en soulignant leur pertinence et leur efficacité dans l'extraction de connaissances à partir de données complexes.**[1]

2 Méthodologie :

2.1 Préparation des Données :

- Les données ont été soigneusement nettoyées pour traiter les valeurs aberrantes. Pour ce faire, nous avons utilisé la moyenne de chaque colonne. On a éliminé les valeurs nulles, supprimé les redondances tant au niveau des colonnes (OC et OM) et des lignes, et appliqué la normalisation Z score pour homogénéiser les données.

2.2 Sélection des Modèles de Clustering :

Deux algorithmes de clustering renommés ont été sélectionnés pour leur capacité à partitionner efficacement des ensembles de données non étiquetées.

1. **K-Means** : K-Means est un algorithme de clustering qui divise un ensemble de données en K clusters compacts en assignant chaque point de données au cluster avec le centre le plus proche.
2. **DBSCAN** : DBSCAN est un algorithme de clustering qui sépare les points de données en clusters basés sur la densité, identifiant les zones de haute densité séparées par des zones de faible densité dans l'espace des caractéristiques.

2.3 Application des Modèles :

Nous avons choisi deux algorithmes de clustering : K-Means et DBSCAN. Voici une explication détaillée de la manière dont chaque algorithme a été appliqué :

1. **L'implémentation de K-Means** :

Algorithm 4 K-Means algorithm

Input: instances, k, max_iterations=100, convergence_threshold=1e-4**Output:** instance_clusters, centroids**if** $k \leq 0$ **then**| **raise** ValueError("Invalid number of clusters or empty dataset")**end**centroids \leftarrow initialize_centroids_kmeans(instances, k)**for** each _ in range(max_iterations) **do**| clusters \leftarrow List of empty lists of size k| **for** i, instance in enumerate(instances) **do**| | distances \leftarrow List of distances from instance to each centroid closest_cluster_index \leftarrow

| | Index of minimum distance in distances Add i to clusters[closest_cluster_index]

| **end**| new_centroides \leftarrow List of centroids calculated from instances in each cluster| variation \leftarrow Calculate variation between new_centroides and centroids| **if** variation < convergence_threshold **then**| | **break**| **end**| centroids \leftarrow new_centroides**end**instance_clusters \leftarrow List of cluster indices for each instance**return** instance_clusters, centroids

L'algorithme K-Means est une méthode de clustering visant à regrouper des instances similaires dans k groupes distincts. La fonction **initialize_centroids_kmeans** utilise l'approche Kmeans++ pour initialiser les centroides de manière optimale. Le processus démarre par l'initialisation aléatoire des centroides, suivie de la répétition d'étapes itératives. À chaque itération, chaque instance est assignée au cluster le plus proche en calculant les distances aux centroides à l'aide de la fonction **calcule_distance_euclidienne**. Ensuite, de nouveaux centroides sont calculés à partir des instances assignées à chaque cluster à l'aide de la fonction **calcule_centroide**. Le critère d'arrêt repose sur la variation de la somme des carrés des distances intra-cluster, comparée à un seuil de convergence défini par **convergence_threshold**. Une fois la convergence atteinte ou après un nombre maximum d'itérations défini par **max_iterations**, les résultats finaux sont renvoyés, comprenant les indices des clusters de chaque instance (**instance_clusters**) et les centroides finaux (**centroides**). Cette approche itérative et progressive de l'algorithme K-Means permet de partitionner les données en clusters distincts en utilisant des centroides pertinents, facilitant ainsi la classification et l'analyse de données.

2. L'implémentation de DBSCAN :

Algorithm 5 Algorithme DBSCAN

Entrée: df - Ensemble de données, eps - Distance maximale entre deux points, min_samples - Nombre minimum de points pour former un cluster

Sortie : labels - Étiquettes de cluster pour chaque point

```
// Initialisation
labels ← Liste de longueur len(df) initialisée à 0s cluster_id ← 0 core_samples ← Liste vide

// Méthode de fit
for i de 0 à len(df) do
    if labels[i] ≠ 0 then
        | continuer
    end
    neighbors ← get_neighbors(df, i) if longueur de neighbors < min_samples then
        | labels[i] ← -1
    end
    else
        | cluster_id ← cluster_id + 1 Ajouter i à core_samples expand_cluster(df, i, neighbors,
        | cluster_id)
    end
end
end
```

L'algorithme DBSCAN (Density-Based Spatial Clustering of Applications with Noise) est une méthode de clustering efficace pour identifier des clusters dans un ensemble de données en se basant sur la densité. La classe **DBSCAN** utilise plusieurs fonctions clés pour effectuer le clustering. La méthode **fit** parcourt chaque point du dataframe, attribuant des étiquettes de cluster en fonction des voisins dans un voisinage défini par **eps** et **min_samples**, utilisant les méthodes **get_neighbors** et **expand_cluster** pour étendre les clusters. La méthode **distance** calcule la distance euclidienne entre deux points. De plus, **calculate_intra_cluster_density** évalue la densité intra-cluster, tandis que **calculate_inter_cluster_density** mesure la densité inter-cluster. Ces fonctions permettent à DBSCAN de détecter des clusters de formes arbitraires et de gérer les points isolés en les marquant comme bruit (-1 dans les étiquettes). Cette approche permet une identification robuste des clusters dans des données de différentes densités, en distinguant efficacement les points de bruit des clusters significatifs.

2.4 Expérimentation en variant les paramètres de k-means et DBSCAN

(a) K-Means :

Notre étude s'est concentrée sur la programmation et l'affinement de l'algorithme de clustering K-means. Ce processus a débuté par une phase d'expérimentation pour identifier les paramètres optimaux, en variant le nombre de clusters (K) et les critères de convergence. Pour chaque combinaison de paramètres, nous avons calculé des métriques de performance telles que l'homogénéité, la complétude, la mesure V et le score de silhouette.

Iterations	Convergence	Homogeneity	Completeness	V_measure	Silhouette
50	0.1	0.2075	0.1604	0.181	0.3901
50	0.01	0.1273	0.0993	0.1115	0.4098
50	0.001	0.066	0.0587	0.0622	0.3921

TABLE 7 – 3 premiers lignes des Résultats de l’Exploration des Hyperparamètres

La heatmap ci-dessus illustre les impacts de différentes itérations sur nos métriques choisies. Nous pouvons observer des variations dans les valeurs de l’homogénéité et de la complétude, ce qui a éclairé notre décision sur le nombre optimal d’itérations à utiliser.

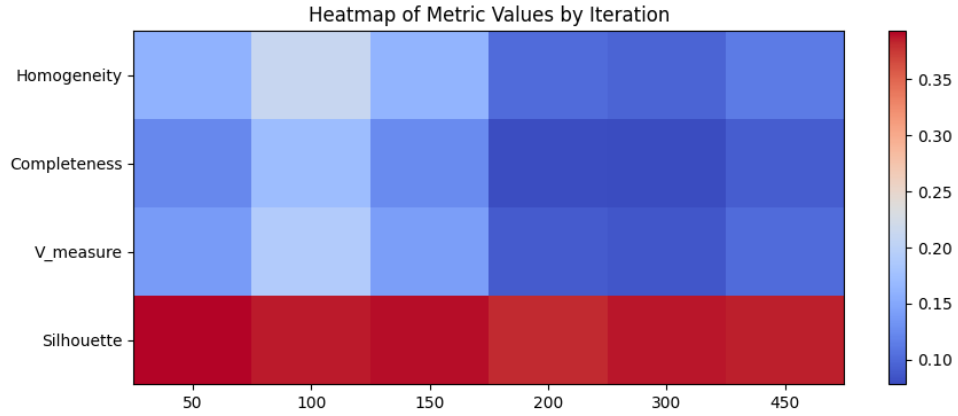


FIGURE 22 – Heatmap pour les Métriques par Itération

De même, la deuxième heatmap montre comment les valeurs de seuil de convergence influencent la performance. Les nuances plus foncées indiquent une meilleure performance, nous guidant vers le choix d’un seuil de convergence qui équilibre précision et efficacité.

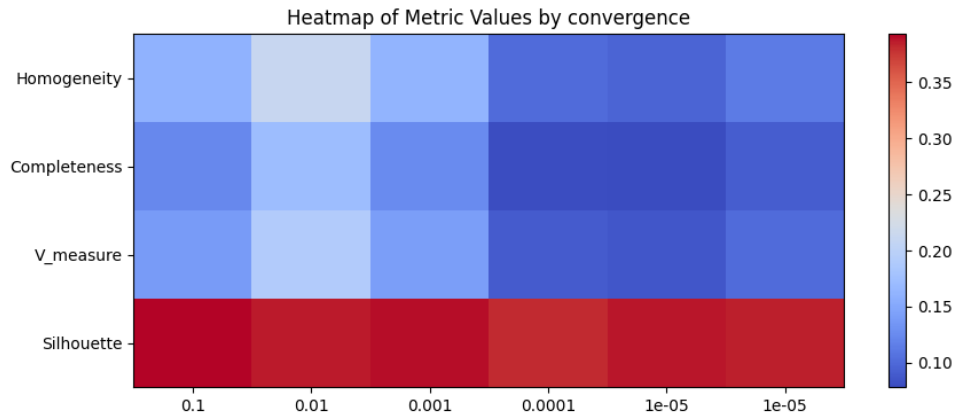


FIGURE 23 – Heatmap pour les Métriques par convergence

Sur la base de ces observations, nous avons sélectionné **un seuil de convergence de 1e-2** et un **nombre d’itérations de 100** comme étant les plus adaptés pour

notre ensemble de données. De plus, la décision de fixer **le nombre de clusters à $K=3$** a été renforcée par l'application de la méthode du coude. Cette méthode implique le calcul de la somme des carrés des distances de chaque point à son centre de cluster attribué pour différents nombres de clusters. Comme l'illustre la courbe ci-dessous, le point où l'inertie commence à diminuer à un rythme plus lent, connu sous le nom de 'point de coude', est clairement visible et indique que $K=3$ est un choix judicieux pour notre nombre de clusters.

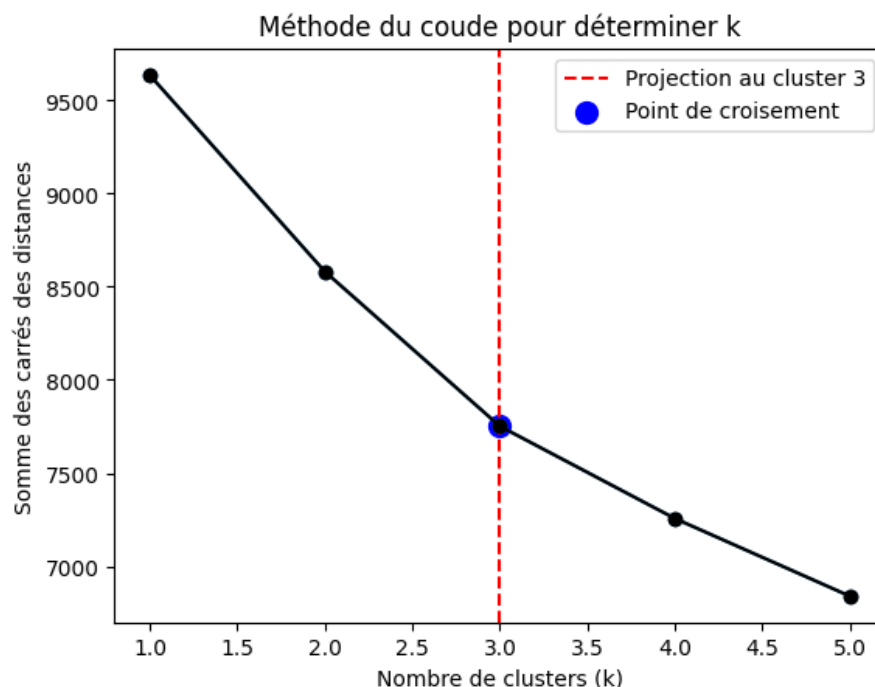


FIGURE 24 – courbe de coude

Le 'point de coude' est marqué par une projection sur l'axe des abscisses, soulignant le moment où augmenter le nombre de clusters n'entraîne plus de réduction significative de l'inertie, et donc $K=3$ est le choix optimal pour notre algorithme K-means.

Evaluation de K-means :

Cette combinaison des paramètres a produit des résultats d'évaluation avec une homogénéité de 0.199, une complétude de 0.1527, et une mesure V de 0.1728, ainsi qu'un indice de silhouette moyen de 0.3883583531974602, suggérant une séparation acceptable entre les clusters.

Le graphique de dispersion PCA ci-dessus démontre la répartition spatiale des clusters formés autour des centroïdes. Les différentes couleurs représentent les clusters distincts, avec les centroïdes marqués par des étoiles, illustrant clairement la ségrégation des données.

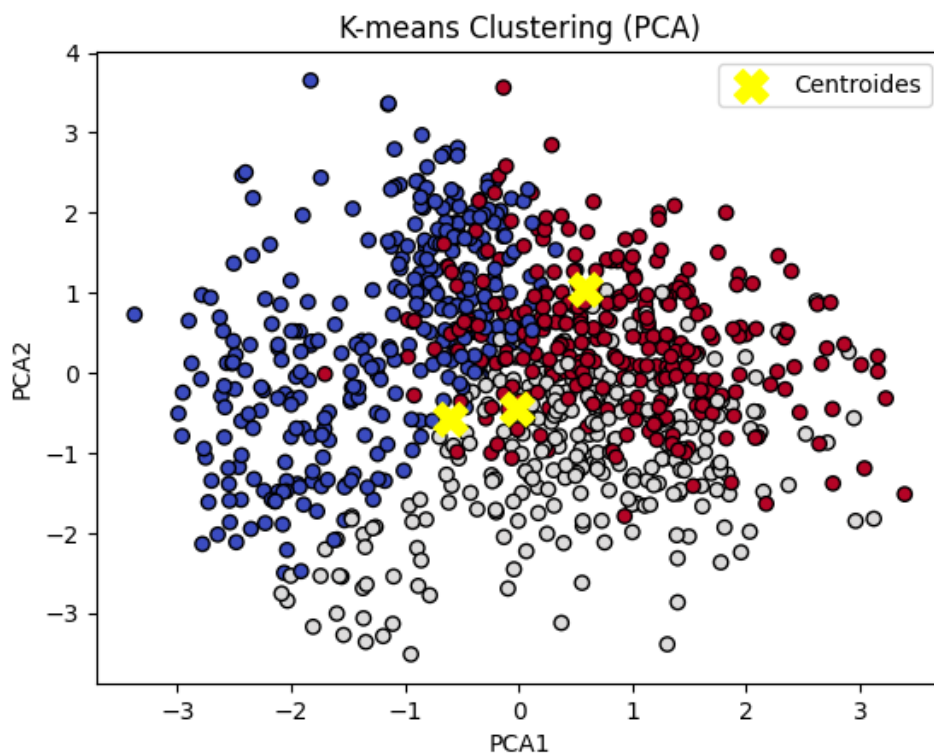


FIGURE 25 – la répartition spatiale des clusters

Enfin, l'histogramme des tailles des clusters révèle une répartition assez uniforme des instances entre les trois groupes créés par l'algorithme k-means. Le premier cluster se distingue par un nombre légèrement supérieur d'instances, tandis que les deux autres clusters affichent des effectifs presque identiques, suggérant que l'algorithme a performé une segmentation qui tend vers une distribution équilibrée des données.

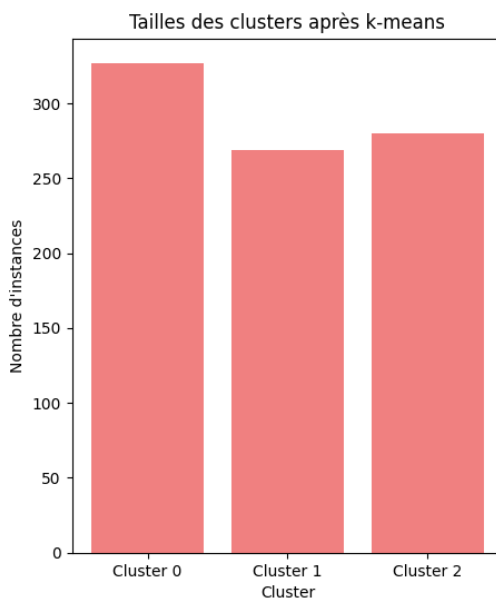


FIGURE 26 – Histogramme des tailles des clusters

(b) DBSCAN Clustering :

Pour l'analyse de clustering DBSCAN, nous avons procédé à une expérimentation avec divers paramètres afin de comprendre leur impact sur la performance du clustering. Le DBSCAN requiert deux paramètres principaux : **eps**, qui est la distance maximale entre deux échantillons pour être considérés comme voisins, et **min_samples**, qui représente le nombre minimal d'échantillons pour former un cluster.

Nous avons itéré sur une plage de valeurs pour **eps** et **min_samples** pour trouver la combinaison qui maximise le score de silhouette, un indicateur de la cohérence interne des clusters.

Le graphique ci-dessus [Figure 27] montre l'effet de la configuration de DBSCAN sur le score de silhouette en fonction d'**eps**. Des valeurs plus élevées d'**eps** semblent augmenter le score de silhouette, indiquant une meilleure définition des clusters.

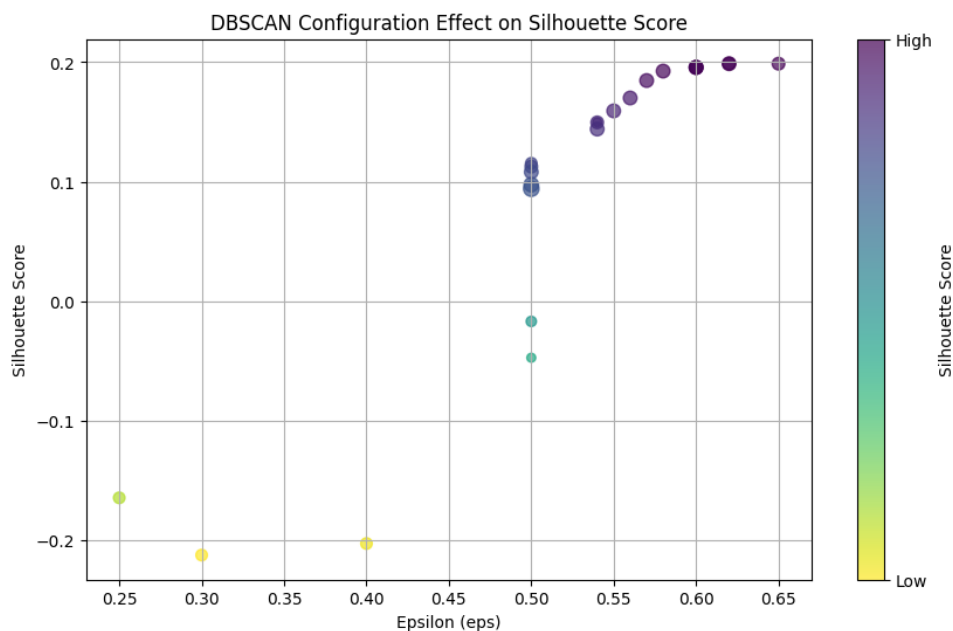


FIGURE 27 – Effet de la configuration DBSCAN sur le score de silhouette en fonction d'eps.

Le graphique suivant [Figure 28] illustre comment le paramètre **min_samples** affecte le score de silhouette. Une valeur optimale se situe là où le score de silhouette est le plus élevé, indiquant des clusters bien séparés et denses.

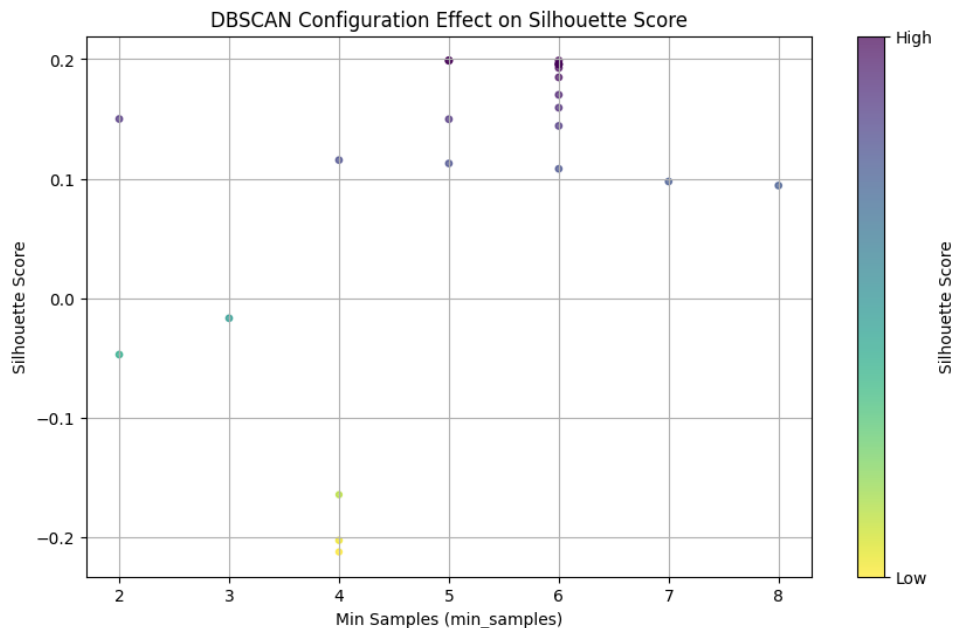


FIGURE 28 – Influence du paramètre min_samples sur le score de silhouette.

Le graphique en surface 3D approfondit cette analyse en montrant l'interaction entre **eps** et **min_samples** et leur effet combiné sur le score de silhouette. Cette visualisation a joué un rôle clé dans le choix des paramètres, nous permettant de sélectionner les valeurs qui optimisent la qualité du clustering.

Impact of EPS and Min Samples on Silhouette Score

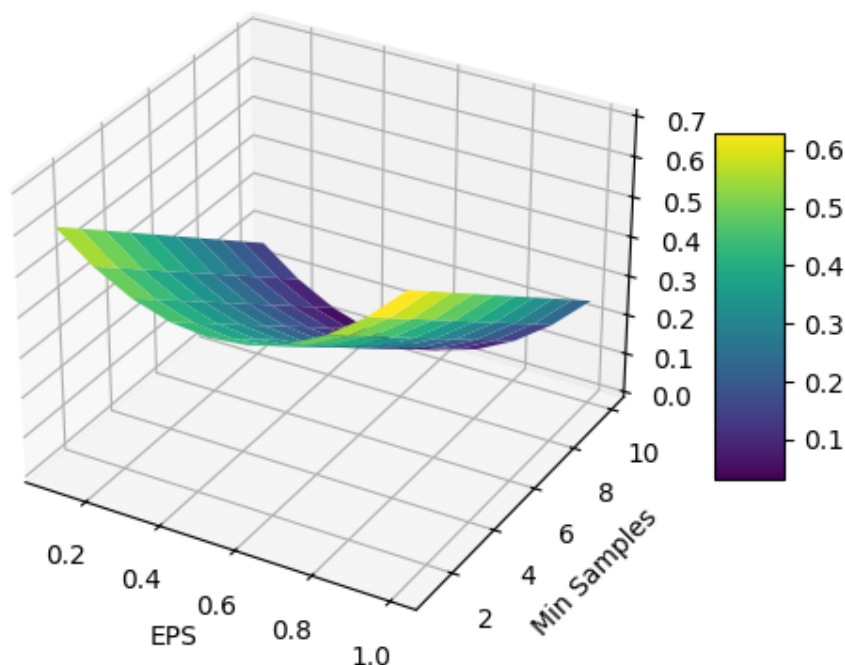


FIGURE 29 – Graphique en surface 3D montrant l'impact de l'interaction entre eps et min_samples sur le score de silhouette.

Après cette analyse, nous avons établi que **eps=0.65** et **min_samples=5** sont les paramètres optimaux pour notre dataset. Avec ces paramètres, le score de silhouette obtenu est de 0.3489, et le nombre de clusters formés est 2, comme l'indiquent le code et les résultats d'évaluation ci-dessous.

```
"Évaluation du score de silhouette et du nombre de clusters pour DBSCAN" avec les valeurs idéale de EPS et Min samples.

eps=0.65
min_samples=5
dbscan = DBSCAN(eps, min_samples)
dbscan.fit(df)
silhouette_avg = silhouette_score(df, dbscan.labels)
num_clusters = len(np.unique(dbscan.labels))
print('eps ', eps, 'min_samples ', min_samples, 'silhouette_score ', silhouette_avg, 'num_clusters ', num_clusters)

eps 0.65 min_samples 5 silhouette_score 0.34891598872938445 num_clusters 2
```

FIGURE 30 – Évaluation du score de silhouette et du nombre de clusters pour DBSCAN

Enfin, la visualisation des clusters DBSCAN à travers une réduction de dimensionnalité PCA montre comment les points de données sont regroupés dans l'espace PCA, soulignant les clusters formés par DBSCAN avec nos paramètres optimaux. Les points colorés représentent les différents clusters identifiés par DBSCAN, illustrant visuellement la séparation et la densité des clusters.

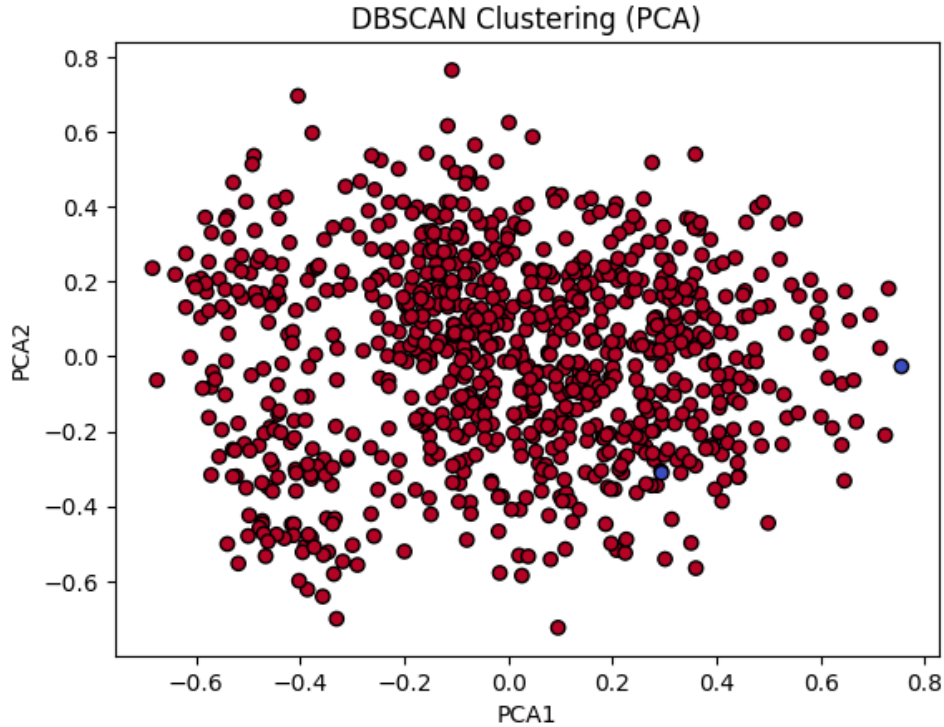


FIGURE 31 – Visualisation DBSCAN Clustering (PCA).

3 Comparaison des Algorithmes

La comparaison des performances des algorithmes de clustering **K-means** et **DBSCAN** s'est appuyée sur l'analyse des métriques de **densité intra-cluster** et **inter-cluster**. La densité intra-cluster, souvent mesurée par l'**inertie**, indique à quel point les instances au sein d'un même cluster sont proches les unes des autres. Pour la densité inter-cluster, une valeur élevée suggère que les clusters sont bien séparés dans l'espace des caractéristiques.

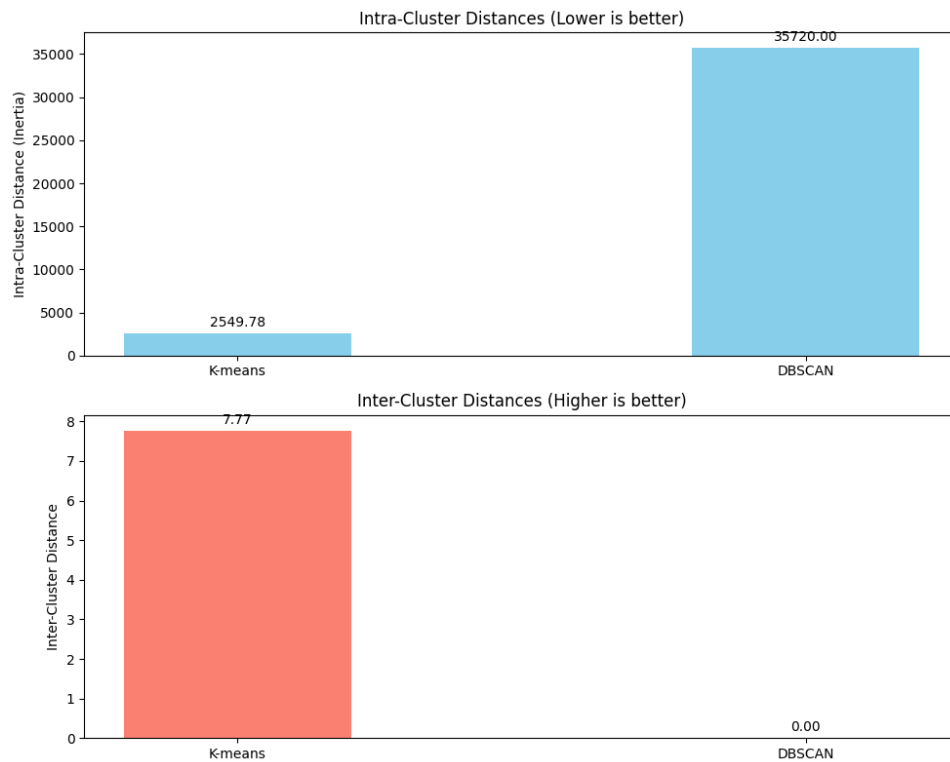


FIGURE 32 – Graphiques en barres des métriques de clustering.

Dans notre analyse de ce graphe en barres [Figure 33], K-means a présenté une **densité intra-cluster considérablement plus faible** que DBSCAN, comme l'indiquent les résultats suivants : K-means avec une inertie de **2549.78** contre **35720.00** pour DBSCAN. Cela signifie que les clusters formés par K-means sont plus cohésifs et serrés par rapport à ceux formés par DBSCAN.

En revanche, pour la **densité inter-cluster**, K-means a montré un résultat significativement meilleur avec une valeur de **7.77**, tandis que DBSCAN a eu une valeur proche de zéro. Cela suggère que K-means est capable de séparer les différents clusters plus distinctement que DBSCAN dans notre jeu de données.

Ces résultats indiquent que **K-means est préférable** lorsque l'on cherche à obtenir des clusters bien définis et distincts les uns des autres. Cependant, **DBSCAN peut être plus approprié** dans des scénarios où les clusters naturels ne sont pas sphériques ou lorsque le bruit et les valeurs aberrantes sont une préoccupation, car il ne force pas les points dans un cluster spécifique et peut traiter le bruit comme une catégorie à part.

En conclusion, le choix entre K-means et DBSCAN dépend fortement de la nature des données et des objectifs de clustering. Pour des données bien séparées où la forme des clusters est approximativement sphérique, K-means est souvent le meilleur choix. Pour des ensembles de données avec des clusters de formes irrégulières ou lorsque la présence de bruit est significative, DBSCAN offre une alternative robuste.

Conclusion Générale

Le traitement analytique des données agricoles a révélé des insights substantiels. Le clustering a mis en exergue des groupements pertinents qui reflètent les différentes qualités de fertilité des sols, tandis que les techniques de classification ont prédit avec précision les niveaux de fertilité à partir des caractéristiques des sols. Les algorithmes testés ont démontré leur utilité, mais aussi leurs limites en fonction des spécificités des données traitées.

Les implications de cette étude sont considérables. L'adoption de ces techniques analytiques peut transformer la manière dont les agriculteurs gèrent leurs cultures et utilisent leurs terres. Pour une mise en application réussie, il est recommandé de poursuivre la recherche dans l'optimisation des modèles et dans l'exploration de données additionnelles qui pourraient affiner encore plus les prédictions.

En conclusion, ce rapport ouvre la voie à de futures recherches et applications pratiques. L'analyse des données, en particulier l'apprentissage automatique, présente un potentiel immense pour l'agriculture intelligente. En continuant à affiner ces techniques et en les intégrant dans des systèmes d'aide à la décision, nous pouvons espérer améliorer significativement la durabilité et l'efficacité de l'agriculture moderne.

Quatrième partie

Annexe

4 Dépendance du code :

Pandas : Une bibliothèque de manipulation et d'analyse de données offrant des structures de données et des opérations pour manipuler des tableaux numériques et des séries temporelles. Pandas est utilisé pour le chargement et le prétraitement des données avant l'application des techniques de clustering.

NumPy : Fondamental pour les calculs scientifiques en Python, NumPy offre un support pour des tableaux multidimensionnels et des matrices, ainsi que des fonctions mathématiques de haut niveau pour opérer sur ces structures de données.

Scikit-learn : Une bibliothèque puissante pour l'apprentissage automatique en Python, qui fournit des implémentations simples et efficaces de nombreux algorithmes de clustering, y compris K-means et DBSCAN, ainsi que des outils pour le calcul des métriques de performance.

Matplotlib et **Seaborn** : Ces bibliothèques de visualisation sont utilisées pour générer des graphiques en barres, des heatmaps et d'autres types de visualisations pour analyser les résultats de clustering et les métriques associées.

Jupyter Notebook : L'environnement de développement interactif utilisé pour la création de notre code. Il permet une intégration transparente du code, des commentaires et des visualisations.

Streamlit : Utilisé pour la création d'interfaces utilisateur interactives, Streamlit permet de transformer rapidement des scripts de data science en applications web partageables.
NB : On utilise la commande "**streamlit run .\interface.py**" pour lancer l'interface.

Bibliographie

- [1] DBSCAN - STEPS. <https://datascientest.com/machine-learning-clustering-dbscan>.
- [2] KNN - Définition. <https://datascientest.com/knn>.